Integrating Representation Learning and Skill Learning in a Human-Like Intelligent Agent

Ph.D. Thesis Proposal
May 21, 2012

Nan Li

Computer Science Department
School of Computer Science
Carnegie Mellon University

Thesis Committee

William W. Cohen (Co-Chair)
Kenneth R. Koedinger (Co-Chair)
Tom Mitchell
Pat Langley (Carnegie Mellon University, Silicon Valley Campus)
Raymond J. Mooney (The University of Texas at Austin)

Abstract

Building an intelligent agent that simulates human learning of math and science could potentially benefit both cognitive science, by contributing to the understanding of human learning, and artificial intelligence, by advancing the goal of creating human-level intelligence. However, constructing such a learning agent currently requires manual encoding of prior domain knowledge; in addition to being a poor model of human acquisition of prior knowledge, manual knowledge-encoding is both time-consuming and error-prone. Previous research has shown that one of the key factors that differentiates experts and novices is their different representations of knowledge. Experts view the world in terms of deep functional features, while novices view it in terms of shallow perceptual features. Moreover, since the performance of learning algorithms is sensitive to representation. The deep features are also important in achieving effective machine learning.

In this work, we propose an efficient algorithm that acquires representation knowledge in the form of "deep features" for specific domains, and demonstrate its effectiveness in the domain of algebra as well as synthetic domains. We integrate this algorithm into a learning agent, SimStudent, which learns procedural knowledge by observing a tutor solve sample problems, and by getting feedback while actively solving problems on its own. We show that learning "deep features" reduces the requirements for knowledge engineering. Moreover, we propose an approach that automatically discovers student models using the extended SimStudent. By fitting the discovered model to real student learning curve data, we show that it is a better student model than human-generated models, and demonstrate how the discovered model may be used to improve a tutoring system's instructional strategy. In future work, we propose to explore other types of perceptual learning, and make use of the acquired representation 1) to further reduce the knowledge engineering effort, 2) to improve learning effectiveness, 3) to better model human student learning.

Contents

1 Introduction

One of the fundamental goals of artificial intelligence is to understand and develop intelligent agents that simulate human-like intelligence. A considerable amount of effort (e.g., [1,41,44]) has been put toward this challenging task. Further, education in the 21^{st} century will be increasingly about helping students not just learn content but become better learners. Thus, we have a second goal of improving our understanding of how humans acquire knowledge and how students vary in their abilities to learn.

1.1 Motivation

To contribute to both goals, there have been recent efforts (e.g., [3, 61, 68, 98]) in developing intelligent agents that model human learning of math, science, or a second language. Although such agents produce intelligent behavior with less human knowledge engineering than before, there remains a non-trivial element of knowledge engineering in the encoding of the prior domain knowledge given to the simulated student agent at the start of the learning process. For example, to build an algebra learning agent, the agent developer needs to provide prior knowledge by coding functions that describe, for instance, how to extract a coefficient or how to add two algebraic terms. Such manual encoding of prior knowledge can be time-consuming and may not correspond with human.

Since real students entering a course do not usually have substantial domain-specific or domain-relevant prior knowledge, it is not realistic in a model of human learning to assume this knowledge is given rather than learned. For example, for students learning about algebra, we cannot assume that they all know beforehand what a coefficient is, or what the difference between a variable term and a constant term is. An intelligent system that models automatic knowledge acquisition with a small amount of prior knowledge could be helpful both in reducing the effort in knowledge engineering intelligent systems and in advancing the cognitive science of human learning.

1.2 Proposed Approach

The goal of this thesis is to build an intelligent agent that is able to learn complex problem solving skills in Science, Technology, Engineering, and Mathematics (STEM) domains from simple demonstrations and feedback. Previous work in this area has developed intelligent agents that acquire complex skills, but the agents' performance often relies heavily on the pre-programmed representations and features specific to the domain. To address this problem, we propose to develop an unsupervised representation/feature learner, and integrate it into a supervised skill-learning agent to improve learning effectiveness of the agent. Hence, unlike normal feature discovery methods that optimize classification or prediction accuracy, the objective of complex skill learning tasks is to jointly optimize learning of perceptual chunks, action chunks or functions, and search control.

There are three main streams of feature construction algorithms. The first category of work has an unsupervised component that learns key features to identify patterns in data (e.g., images), and then a supervised learning process that makes use of these features to optimize some objective function. An example of such work is deep belief networks [29].

Other work takes a joint learning strategy to build latent variable discriminative models such as supervised LDA [9]. A third branch in feature construction is supervised feature induction, which searches for new features to optimize an objective function.

Our feature/representation learner falls into the first category, where it focuses on building a generative model, $\mathcal{G}$, that best captures the distribution among observations (e.g., algebraic expressions), $\mathcal{R}$, which approximates maximum likelihood estimate (MLE) of $p(\mathcal{R}|\mathcal{G})$. Although not trained by directly optimizing the learning effectiveness of the intelligent agent, the philosophy behind this strategy is that if we could correctly model these observations (e.g., the right parse structures for the expressions), the acquired representation should contain useful features that aid the skill learning process, and yield faster learning.

The idea of our representation learner comes from previous work in cognitive science [16, 17], which showed that one of the key factors that differentiates experts and novices in a field is their different prior knowledge of world state representation. Experts view the world in terms of deep functional features (e.g., coefficient and constant in algebra), while novices only view in terms of shallow perceptual features (e.g., integer in an expression). Deep feature learning is a major component of human expertise acquisition, but has not received much attention in AI. Learning deep features changes the representation on which future learning is based and, by doing so, improves future learning. However, how these deep features are acquired is not clear. Therefore, we have recently developed a learning algorithm that acquires deep features automatically with only domain-independent knowledge (e.g., what is an integer) as input [52]. We evaluated the effectiveness of the algorithm in learning deep features, but not its impact on future skill learner.

In order to evaluate how the deep feature learner could affect future learning of an intelligent agent, we further integrated this deep feature learning algorithm into *SimStudent* [61], an agent that learns problem-solving skills by examples and by feedback on performance. The original SimStudent relies on a hand-engineered representation that encodes an expert representation given as prior knowledge. This limits its ability to model novice students. Integrating the deep feature learner into the original SimStudent both reduces the amount of engineering effort and builds a better model of student learning.

We show that the extended SimStudent with better representation learning performs much better than the original SimStudent when neither of them are given domain-specific knowledge. Furthermore, we also show that even compared to the original SimStudent with the domain-specific knowledge, the extended SimStudent is able to learn nearly as well without being given domain-specific knowledge. In addition, we use the extended SimStudent to automatically discover models of real students, and show that the discovered models are better student models than human-generated models.

In future work, we propose to continue exploring other types of representation learning, and make use of the acquired representation 1) to further reduce the knowledge engineering effort, 2) to improve learning effectiveness, 3) to better model human student learning.

1.3 Expected Contribution

To summarize, the main contributions of this work are two-fold. By integrating representation learning into skill learning, 1) we reduce the amount of knowledge engineering effort required in constructing an intelligent agent; 2) we get a better modeling of human learning

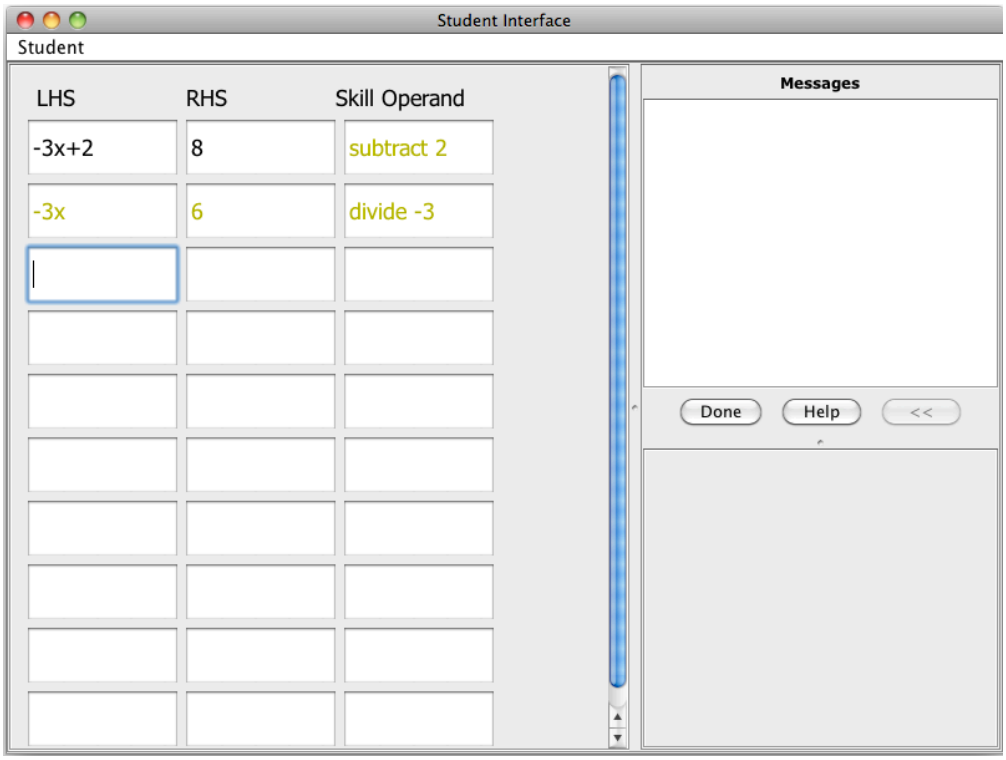


Figure 2.1: A simple interface to tutor SimStudent in equation solving.

behavior. Note that rather than duplicating how the human brain works, our focus of this work is to build a system that behaves like human students.

2 Prior Work

In this section, we start with a brief review of SimStudent. Next, we present the deep feature representation learning algorithm together with its evaluation results. Next, we describe how to integrate the representation learner into SimStudent, and illustrate the algorithm with an example in algebra. After that, we present experimental results for both the original SimStudent and the extended SimStudent trained with problem sets used by real students in learning algebra, and show that the extended SimStudent is able to achieve performance comparable to the original SimStudent without requiring domain-specific knowledge as input. We present a method for using SimStudent to automatically discover student models, and show that the student model discovered by the extended SimStudent is better than the human-generated models.

2.1 A Brief Review of SimStudent

SimStudent is an intelligent agent that inductively learns skills to solve problems from demonstrated solutions and from problem solving experience. It is an extension of programming by demonstration [48] using inductive logic programming [67] as one of its underlying learning

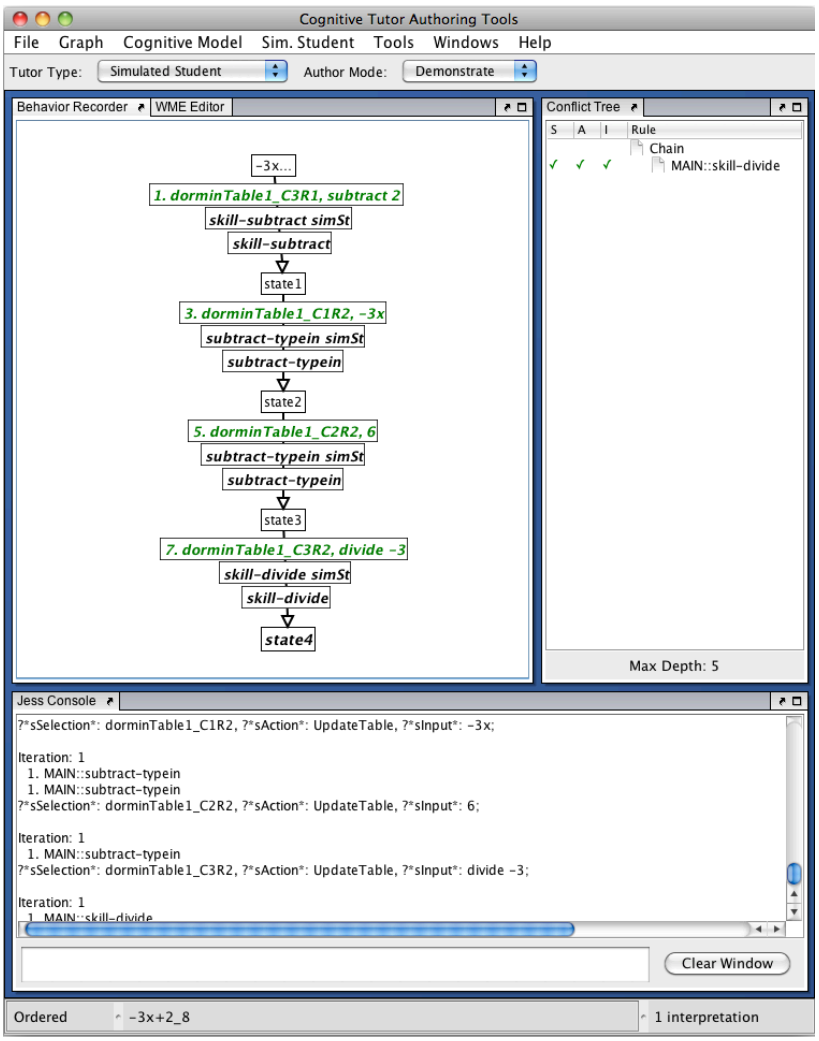


Figure 2.2: The interface that shows how SimStudent traces each demonstrated step and learns production rules.

techniques. Figure 2.1 and 2.2 are screenshots of SimStudent learning to solve algebra equations. Figure 2.1 is an interface used to teach SimStudent equation solving, and Figure 2.2 shows how SimStudent keeps track of the demonstrated steps and acquires skill knowledge based on them. In this paper, we will use equation solving as an illustrative domain to explain the learning mechanisms. However, the learning algorithms are domain general. In fact, SimStudent has been used and tested across various domains, including multi-column addition, fraction addition, stoichiometry, and so on. In the rest of this subsection, we will briefly review the learning mechanism of SimStudent. For full details, please refer to [53].

2.1.1 Learning Task

SimStudent is given a set of (ideally simple) *feature predicates* and a set of (ideally simple) *operator functions* as prior knowledge before learning. Each feature predicate is a boolean

function that describes relations among objects in the domain. For example, *(has-coefficient -3x)* means *-3x* has a coefficient.

Operator functions specify basic functions (e.g., add two numbers, get the coefficient) that SimStudent can apply to aspects of the problem representation. Operator functions are divided into two groups, domain-independent operator functions and domain-specific operator functions. Domain-independent operator functions can be used across multiple domains, and tend to be simpler (like standard operations on a programming language). Examples of such operator functions include adding two numbers, *(add 1 2)* or copying a string, *(copy -3x)*. These operator functions are not only useful in solving equations, but can also be used in other domains such as multi-column addition and fraction addition. Because these domain-general functions are involved in domains that are acquired before algebra, we can assume that real students know them prior to algebra instruction. Because these domain-general functions can be used in multiple domains, there is a potential engineering benefit in reducing or eliminating a need to write new operator functions when applying SimStudent to a new domain. Domain-specific operator functions, on the other hand, are more complicated functions, such as getting the coefficient of a term, *(coefficient -3x)*, or adding two terms. Performing such operator functions implies some domain expertise that real students are less likely to have.

Domain-specific operator functions tend to require more knowledge engineering or programming effort than domain-independent operator functions. For example, compare the "add" domain-independent operator function with the "add-term" domain-specific operator function. Adding two numbers is one step among the many steps in adding two terms together (i.e., parsing the input terms into sub-terms, applying an addition strategy for each term format, and concatenating all of the sub-terms together).

Note that operator functions are different from *operators* in traditional planning systems, operator functions have no explicit encoding of preconditions and may not produce correct results when applied in context. Thus, SimStudent is different from traditional planning algorithms, which can engage on speed-up learning. SimStudent engages in knowledge-level learning [69], and inductively acquires complex reasoning rules. These rules are represented as *production rules*, which we will explain later.

During the learning process, given the current state of the problem (e.g., *-3x = 6*), SimStudent first tries to find an appropriate production rule that proposes a plan for the next step (e.g., *(coefficient -3x ?coef) (divide ?coef)*). If it finds one and receives positive feedback, it continues to the next step. If the proposed next step is incorrect, negative feedback is given, and if SimStudent has no other alternatives, a correct next step demonstration is provided. SimStudent will attempt to modify or learn production rules accordingly. Although other feedback mechanisms are also possible, in our case, the feedback is given by an existing automatic cognitive tutor [38], which has been used to teach real students. For each demonstrated step, the tutor specifies 1) *perceptual information* (e.g., *-3x* and *6* for *-3x = 6*) from a graphical user interface (GUI) showing where to find information to perform the next step, 2) *a skill label* (e.g., divide) corresponding to the type of skill applied, 3) *a next step* (e.g., *(divide -3)* for problem *-3x = 6*). This simulates the more limited information available to real students. Taken together, the three pieces of information form an *example action record* indexed by the skill label, $R=\langle label, \langle percepts, step\rangle\rangle$. In the algebra example, an example action record is $R=\langle divide, \langle (-3x, 6), (divide\ -3)\rangle\rangle$. For each incorrect next step

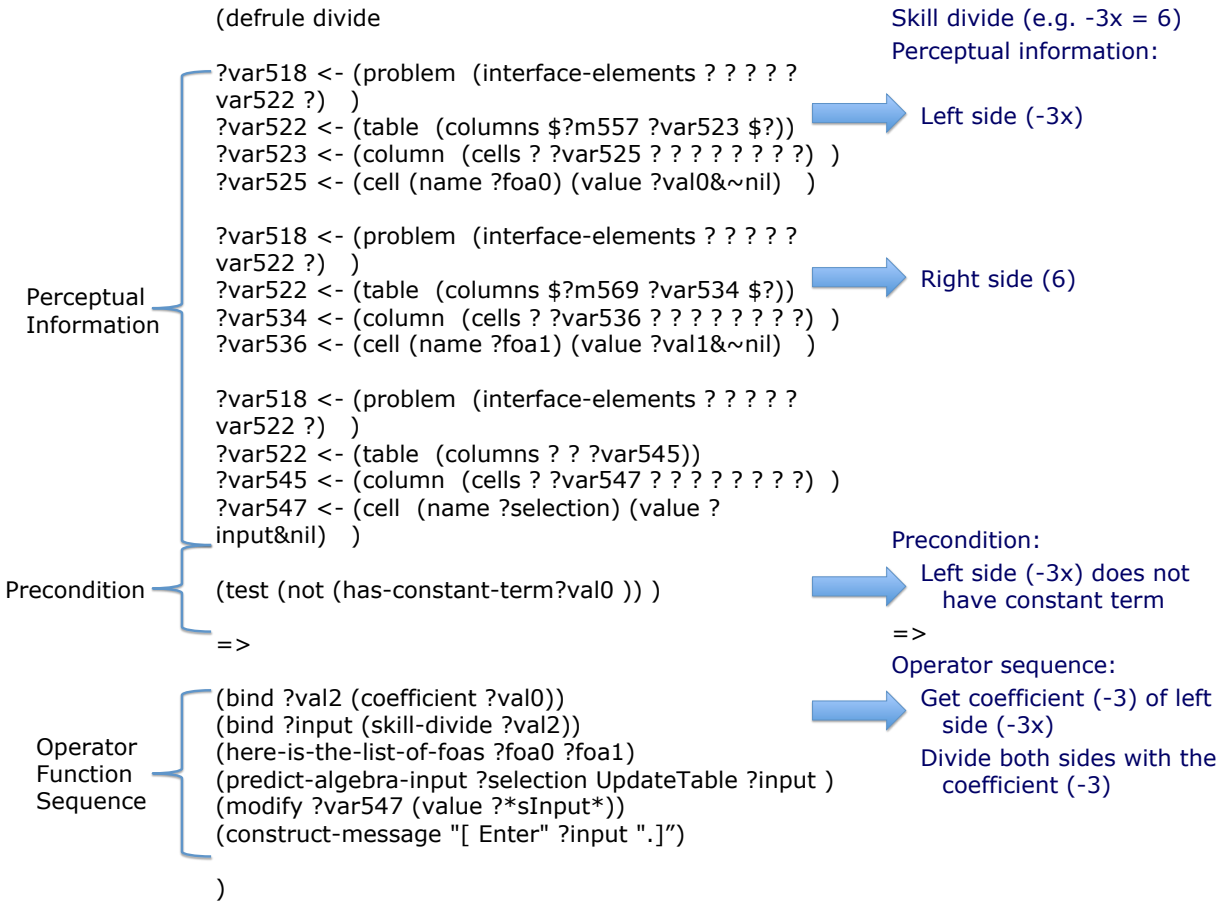


Figure 2.3: A production rule for divide.

proposed by SimStudent, an example action record is also generated as a negative example. During learning, SimStudent typically acquires one production rule for each skill label, l, based on the set of associated (both positive and negative) example action records gathered up to the current step, $\mathcal{R}_l = (R_1, R_2, ..., R_n)$ (where $R_i.label = l$).

In summary, since we would like to model how real students are tutored, the learning task presented to SimStudent is challenging. First, the total number of world states is large. In equation solving, for instance, there are infinite variety of algebraic expressions that can be entered and there are many possible alternative solution strategies. Second, the operator functions given as prior knowledge do not encode any preconditions (neither for applicability nor for search control) or postconditions. Last, the semantics of a demonstrated step is only partially observable. It usually takes more than one operator function to move from one observed state to the next observed state. Correct intermediate outputs of operator functions are unobservable to SimStudent. Taken together, the learning task SimStudent is facing is learning skill knowledge within infinite world states given incomplete operator function descriptions and partially observable states.

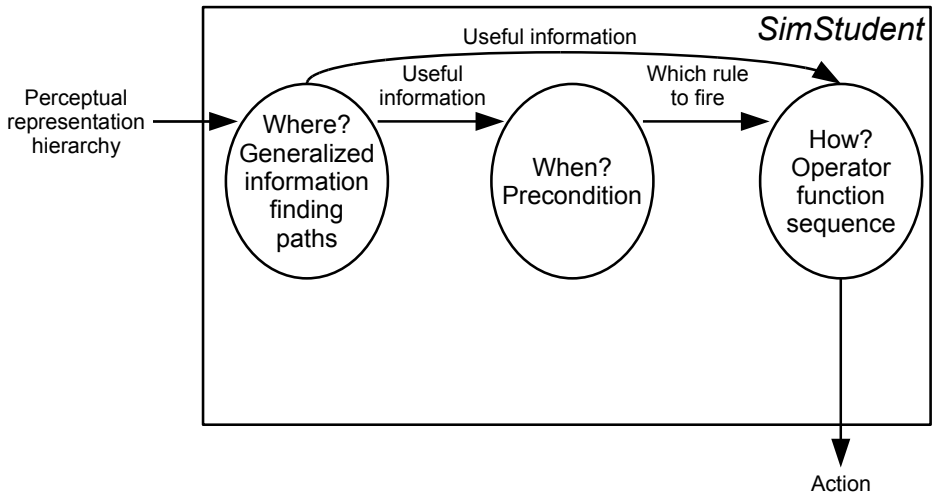


Figure 2.4: A diagram about how SimStudent reacts with the environment. Solid lines show the information that flows through components during execution. Information about the training examples is not presented.

2.1.2 Production Rules

The output of the learning agent is represented as production rules [1, 41]. The left side of Figure 2.3 shows an example of a production rule learned by SimStudent with a simple English description shown on the right. A production rule indicates "where" to look for information in the interface (perceptual information), "how" to change the problem state (an operator function sequence), and "when" to apply a rule (a set of features indicting the circumstances under which performing the how-part will be useful). For example, the rule to "divide both sides of *-3x=6* by *-3*" shown in Figure 2.3 can be read as "given a left-hand side (i.e., *-3x*) and a right-hand side (6) of an equation, when the left-hand side does not have a constant term, then get the coefficient of the term on the left-hand side and write "divide" followed by the coefficient." The perceptual information part represents paths to identify useful information from the GUI. The precondition (just before "⇒" in Figure 2.3) includes a set of feature tests representing desired conditions in which to apply the production rule. The last part (after "⇒" in Figure 2.3) is the operator function sequence which computes what to output in the GUI.

During execution, as shown in Figure 2.4, SimStudent receives perceptual information from the environment as a hierarchy. The where part finds the useful information from this hierarchy. Next, the when part uses the useful information to decide which production rule to fire. The selected production rule will generate an action that SimStudent is going to execute in the world decided by the how part of the production rule.

2.1.3 Learning Mechanisms

With all the challenges presented, we have developed three learning mechanisms in SimStudent to acquire the three parts of the production rules as shown in Figure 2.4 [61], where each learning component models one aspect of problem-solving skill acquisition. The first component is a perceptual learner that learns the where-part of the production rule by finding

paths to identify useful information in the GUI. The elements in the interface are typically organized in a tree structure. For example, the table node has columns as children, and each column has multiple cells as children. The percepts specified in the above production rule are cells associated with the sides of the algebra equation, which are *Cell 11* and *Cell 21* in this case. Hence, the perceptual learner's task is to find the right paths in the tree to reach the specified cell nodes. There are two ways to reach a percept node in the interface: 1) by the exact path to its exact position in the tree, or 2) by a generalized path to a set of GUI elements that may have a specific relationship with the GUI element where the next step is entered (e.g., cells above next step). A generalized path has one or more levels in the tree that are bound to more than one node. For example, a cell in the second column and the third row, *Cell 23*, can be generalized to any cell in the second column, *Cell 2?*, or any cell in the table, *Cell ??*. In the example shown in Figure 2.3, the production rule has an over-specific where-part that produces a next step only when the sides of the current step are in row two. The learner searches for the least general path in the version space formed by the set of paths to training examples [63]. This process is done by a brute-force depth-first search. For example, if only given the example *-3x=6* in row two, the production rule learned as shown in Figure 2.3 has an over-specific where-part. If given more examples in other rows (e.g., *4x=12* in row three), the where-part will be generalized to any row in the table.

The second part of the learning mechanism is a feature test learner that learns the when-part of the production rule by acquiring the precondition of the production rule using the given feature predicates. The acquired preconditions should contain information about both applicability (e.g., getting a coefficient is not applicable to the term *3x+5*) and search control (e.g., it is not preferred to add *5* to both sides for problem *-3x = 6*). The feature test learner utilizes FOIL [74], an inductive logic programming system that learns Horn clauses from both positive and negative examples expressed as relations. FOIL is used to acquire a set of feature tests that describe the desired situation in which to fire the production rule. For each rule, the feature test learner creates a new predicate that corresponds to the precondition of the rule, and sets it as the target relation for FOIL to learn. The arguments of the new predicate are associated with the percepts. Each training action record serves as either a positive or a negative example for FOIL. For example, *(precondition-divide ?percept$_1$?percept$_2$)* is the precondition predicate associated with the production rule named "divide". *(precondition-divide -3x 6)* is a positive example for it. The feature test learner computes the truthfulness of all predicates bound with all possible permutations of percept values, and sends it as input to FOIL. Given these inputs, FOIL will acquire a set of clauses formed by feature predicates describing the precondition predicate.

The last component is an operator function sequence learner that acquires the how-part of the production rule. For each positive example action record, R_i, the learner takes the percepts, R_i.*percepts*, as the initial state, and sets the step, R_i.*step*, as the goal state. We say an operator function sequence *explains* a percepts-step pair, $\langle R_i.percepts,\ R_i.step\rangle$, if the system takes R_i.*percepts* as an initial state and yields $step_i$ after applying the operator functions. For example, if SimStudent first receives a percepts-step pair, $\langle$*(2x, 2), (divide 2)*$\rangle$, both the operator function sequence that directly divides both sides with the right-hand side (i.e., *(divide ?2)*), and the sequence that first gets the coefficient, and the divides both sides with the coefficient (i.e., *(coefficient 2x ?coef) (divide ?coef)*) are possible explanations for

Table 2.1: Probabilistic context free grammar for coefficient in algebra

Terminal symbols: $-, x$;
Non-terminal symbols: $Expression, SignedNumber,$
$Variable, MinusSign, Number$;
$Expression \rightarrow 1.0, \quad [SignedNumber] \quad Variable$
$Variable \rightarrow 1.0, \quad x$
$SignedNumber \rightarrow 0.5, \quad MinusSign \quad Number$
$SignedNumber \rightarrow 0.5, \quad Number$
$MinusSign \rightarrow 1.0, \quad -$

the given pair. Since we have multiple example action records for each skill, it is not sufficient to find one operator function sequence for each example action record. Instead, the learner attempts to find a shortest operator function sequence that explains all of the ⟨*percepts, step*⟩ pairs using iterative-deepening depth-first search within some depth-limit. As in the above example, since *(divide ?2)* is shorter than (i.e., *(coefficient 2x ?coef) (divide ?coef)*), SimStudent will learn this operator function sequence as the how-part. Later, it meets another example, *-3x=6*, and receives another percepts-step pair, ⟨*(-3x, 6), (divide -3)*⟩. The operator function sequence that divides both sides with the right-hand side is not a possible explanation any more. Hence, SimStudent modifies the how-part to be the longer operator function sequence *(coefficient ?rhs ?coef) (divide ?coef)*.

Last, although we said that SimStudent tries to learn one rule for each label, when a new training action record is added, SimStudent might fail to learn a single rule for all example action records (e.g., no operator function sequence is found that explains all percepts-step pairs including the new one). In that case, SimStudent learns a separate rule just for the last example action record. This breaking a single production rule into a pair of disjuncts effectively splits the example action records into two clusters. Later, for each new example action record, SimStudent tries to acquire a rule for each of the example clusters plus the new example action record.

2.2 Deep Feature Representation Learning

Having reviewed SimStudent's production rule learning mechanisms, we move to a discussion of representation knowledge acquisition as deep feature learning. As mentioned above, deep feature learning is important both for human knowledge acquisition, and in achieving effective machine learning. We carefully examined the nature of deep feature learning in algebra equation solving, and discovered that it could be modeled as an unsupervised grammar induction problem given observational data (e.g., expressions in algebra). Expressions can be formulated as a context free grammar and deep features are modeled as grammar non-terminal symbols in particular positions in a grammar rule. Table 2.1 illustrates a portion of a grammar for algebra expressions and the modeling of the deep feature "coefficient" as a non-terminal symbol in one of the grammar rules, as indicated by the square brackets (i.e., $[SignedNumber]$).

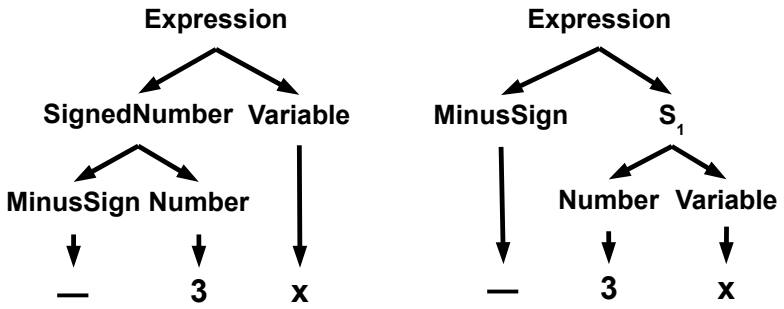


Figure 2.5: Correct and incorrect parse trees for $-3x$.

Viewing feature learning tasks as grammar induction provides a general explanation of how experts acquire perceptual chunks [16, 36] and specific explanations for novice errors. In this account, some novice errors are the result of acquiring the wrong grammar for the task domain. Let us use the *-3x* example again. The correct grammar shown in Table 2.1 produces the correct parse tree shown on the left in Figure 2.5. A novice, however, may acquire different grammar rules (e.g., because of plausible lack of experience with negative numbers) and these result in the incorrect parse tree shown on the right of Figure 2.5. Instead of grouping - and *3* together, this grammar groups *3* and *x* first, and thus mistakenly considers *3* as the coefficient. In fact, a common strategic error students make in a problem like *-3x=12* is for the student to divide both sides by *3* rather than *-3* [55]. Based on observations like these, we built a deep feature learner by extending an existing probabilistic context free grammar (pCFG) learner [56] to support feature learning and transfer learning. Note that the deep feature learner is domain general. It currently supports domains where student input can be represented as a string of tokens, and can be modeled with a context-free grammar (e.g., algebra, chemistry, natural language processing).

2.2.1 A Brief Review of the Grammar Learner

Before introducing the deep feature acquisition algorithm, we first briefly review the pCFG learner [56] it is based on. The pCFG learner is a variant of the inside-outside algorithm [47] that acquires a probabilistic context-free grammar (pCFG). The input to the pCFG learner is a set of observation sequences, $\mathcal{O}$. Each sequence is a string of tokens directly from user input. The output is a pCFG that can generate all input observation sequences with high probabilities. The system consists of two parts, a greedy structure hypothesizer (GSH), which creates non-terminal symbols and associated grammar rules, as needed, to cover all the training examples, and a Viterbi training step, which iteratively refines the probabilities of the grammar rules.

Greedy Structure Hypothesizer (GSH): Pseudo code for the GSH algorithm is shown in algorithm 1. GSH creates context-free grammar in a bottom-up fashion. It starts by initializing the rule set S to rules associated with terminal characters (e.g., *3* and *x* in *3x*) in the observation sequences, O. Next the algorithm (line 4) detects whether there are possible recursive structures embedded in the observation sequence by looking for repeated symbols.

Input: Observation Sequence Set O.
S := terminal symbol grammar rules;
while *not-all-sequences-are-parsable(O, S)* **do**
  if *has-recursive-rule(O)* **then**
    s := generate-recursive-rule(O);
  else
    s := generate-most-frequent-rule(O);
  end
  $S := S + s$;
  O := update-plan-set-with-rule(O, S);
end
S = initialize-probabilities(S); **return** S

Algorithm 1: GSH constructs an initial set of grammar rules, S, from observation sequences, O.

If so, the algorithm learns a recursive rule for them. If the algorithm fails to find recursive structures, it starts to search for the character pair that appears in the plans most frequently (line 6), and constructs a grammar rule for the character pair. To build a non-recursive rule, the algorithm will introduce a new symbol and set it as the head of the new rule. After getting the new rule, the system updates the current observation set O with this rule by replacing the character pairs in the observations with the head of the rule (line 9).

After learning all the grammar rules, the structure learning algorithm assigns initial probabilities to these rules. If there are k grammar rules with the same head symbol, then each of them are assigned the probability $\frac{1}{k}$. To break ties among grammar rules with the same head, GSH adds a small random number to each probability and normalizes the values again. This output of GSH is a redundant set of grammar rules, which is sent to the Viterbi training phase.

Refining Schema Probabilities – Viterbi Training Phase: The probabilities associated with the initial set of rules generated by the GSH phase are tuned by a Viterbi training algorithm. It considers the parse trees T associated with each observation sequence as hidden variables. Each iteration involves two steps.

In the first step, the algorithm computes the most probable parse tree for each observation example using the current rules. Any subtree of a most probable parse tree is also a most probable parse subtree. Therefore, for each observation sequence, the algorithm builds the most probable parse tree in a bottom-up fashion until reaching the start symbol. After getting the parse trees for all observation examples, the algorithm moves on to the second step. In this step, the algorithm updates the selection probabilities associated with the grammar rules. For a grammar rule with head a_i, the new probability of being chosen is simply the total number of times that schema appears in the Viterbi parse trees divided by the total number of times a_i appears in the parse trees. (This learning procedure is a fast approximation of expectation-maximization [23], which approximates the posterior distribution of trees given parameters by the single MAP hypothesis.) After finishing the second step, the algorithm starts a new iteration until convergence. The output of the algorithm is a set of probabilistic grammar rules.

2.2.2 Feature Learning

Having reviewed Li et al.'s pCFG learning algorithm, we are ready to describe how it is extended to support deep feature learning without SimStudent. The input of the system is a set of pairs such as ⟨*-3x, -3*⟩, where the first element is the input to a feature extraction mechanism (e.g., coefficient), and the second is the extraction output (e.g., *-3* is the coefficient of *-3x*). The output is a pCFG with a non-terminal symbol in one of the rules set as the target feature (as shown by [*SignedNumber*] in Table 2.1). To produce this output, the deep feature learner uses the pCFG learner to produce a grammar, and then searches for non-terminal symbols that correspond to the example extraction output (e.g., the *-3* in *-3x*). The process is done in three steps.

The system first builds the parse trees for all of the observation sequences based on the acquired rules. For instance, in algebra, suppose we have acquired the pCFG shown in Table 2.1. The associated parse tree of *-3x* is shown at the left side of Figure 2.5. Next, for each sequence, the learner traverses the parse tree to identify the non-terminal symbol associated with the target feature extraction output, and the rule to which the non-terminal symbol belongs. In the case of our example, the non-terminal symbol is *SignedNumber*, the associated feature extraction output is *-3*, and the rule is *Expression → 1.0, SignedNumber Variable*. For some of the sequences, the feature extraction output may not be generated by a single non-terminal symbol, which happens when the acquired pCFG does not have the right structure. For example, the parse tree shown in the right side of Figure 2.5 is an incorrect parse of *-3*, and there is no non-terminal symbol associated with *-3*. In this case, no non-terminal symbol is associated with the target feature for the current sequence. Last, the system records the frequency of each symbol rule pair, and picks the pair that matches the most training records as the learned feature. For instance, if most of the input records match with *SignedNumber* in *Expression → 1.0, SignedNumber Variable*, this symbol-rule pair will be considered as the target feature pattern.

After learning the feature, when a new problem comes, the system will first build the parse tree of the new problem based on the acquired grammar. Then, the system recognizes the subsequence associated with the feature symbol from the parse tree, and returns it as the target feature extraction output (e.g., *-5* in *-5x*).

2.2.3 Transfer Learning for Deep Feature Learning

In order to achieve effective learning, we further extended the feature learner to support transfer learning within the same domain and across domains. Different grammars sometimes share grammar rules for some non-terminal symbols. For example, both the grammar of equation solving and the grammar of integer arithmetic problems should contain the sub-grammar of signed number. We extended the feature learning algorithm to transfer solutions to common sub-grammars from one task to another. Note that the tasks can be either from the same domain (e.g. learning what is an integer, and learning what is a coefficient), or from different domains (e.g. learning what is an integer, and learning what is a chemical formula). We consider two learning protocols: one in which the tutor provides hints to a shared grammar by highlighting subsequences that should be associated with a non-terminal symbol; and one in which the shared grammar is present, but no hints are provided. For transfer

learning with sub-grammar hints, we applied what we will call a *feature focus mechanism* to the acquisition process. For transfer learning without sub-grammar hints, we extended the system to make use of grammar rule application frequencies from previous tasks to guide future learning, as explained below.

Explicitly Labeled Common Sub-grammars: We first consider the situation where SimStudent's tutor provides a hint toward a shared sub-grammar (the deep feature). In the original learning algorithm, during the process of grammar induction, the learner acquires some grammar that generates the observation sequences, without differentiating potential feature subsequences (e.g. coefficients or constant terms) from other subsequences in the training examples. It is possible that two grammars can generate the same set of observation sequences, but only one grammar has the appropriate feature symbol embedded in it. We cannot be sure that the original learner will learn the right one.

However, it may be reasonable to assume that a tutor explicitly highlights example subsequences as targeted features as in a teacher giving examples of coefficient by indicating that *-3* is the coefficient of *-3x* and *-4* is the coefficient of *-4x*. With this assumption, the deep feature learner can focus on creating non-terminal symbols for such feature subsequences. We developed this *feature focus mechanism* as follows. First, we call one copy of the original learner to learn the subgrammar for the deep feature. That is, we extract all the feature subsequences from training sequences, and then learn a sub-grammar for it. We then replace the feature subsequence with a special *semantic terminal symbol*, and invoke the original learner on this problem. Since this semantic terminal symbol is viewed as a terminal character in this phase of learning, it must be properly embedded in the observation sequence. Finally, the two grammars are combined, and the semantic terminal is relabeled as a *non-terminal symbol* and associated with the start symbol for the grammar for the feature.

Learning and Transfer of Common Sub-grammars without Hints: Aiding transfer learning by providing hints for common sub-grammars requires extra work for the tutor. A more powerful learning strategy should be able to transfer knowledge without adding more work for the tutor. Therefore, we considered a second learning protocol, where the shared grammar is present, but no hints to it are provided. An appropriate way of transferring previously acquired knowledge to later learning could improve the speed and accuracy of that later learning. The intuition here is that the perceptual chunks or grammar acquired with whole-number experience will aid grammar acquisition of negative numbers which, in turn, will aid algebra grammar acquisition. Our solution involves transferring the acquired grammar, including the application frequency of each grammar rule, from previous tasks to future tasks.

More specifically, during the acquisition of the grammar in previous tasks, the learner records the acquired grammar and the number of times each grammar rule appeared in a parse tree. When faced with a new task, the learning algorithm first uses the existing grammar from previous tasks to build the smallest number of most probable parse trees for the new records. This process is done in a top-down fashion. For each sequence/subsequence, the algorithm first tries to see whether the given sequence/subsequence can be reduced to a single most probable parse tree. If it succeeds, the algorithm returns; if it fails, the algorithm separates the sequence/subsequence into two subsequences, and recursively calls itself. After building the least number of most probable parse trees for the training subsequences, the

system switches to the original GSH and acquires new rules based on the partially parsed sequences.

For example, if the grammar learner acquired what is a signed number (e.g. *-3*) in a previous task, when faced with a new task of learning what is a term (e.g. *-3x*), the learner first tries to build a parse tree for the whole term (e.g. *-3x*). But it fails because the grammar for signed number can only build the parse trees for some subsequence (e.g. *-3* in *-3x*). Nevertheless, the grammar learner does get some partially parsed sequences (e.g. the partial reduced sequence for *-3x* is *SignedNumber x*), and calls the original grammar learner on these partially parsed sequences.

During the Viterbi training phase, the learning algorithm estimates rule frequency using a Dirichlet distribution based on prior tasks; that is, it adds the applied rule frequency associated with the training problems of the current task to the recorded frequency from previous tasks. Note that it is possible that after acquiring new rules with new examples, in the Viterbi training phase, the parse trees for the training examples in the previous tasks have changed, and the recorded frequencies are no longer accurate, so this is not equivalent to combining the examples from the old task with the examples of the new task. By recording only the frequencies, instead of rebuilding the parse trees for all previous training examples in each cycle, we save both space and time for learning.

Having acquired the grammar for deep features, when a new problem is given to the system, the learner will extract deep features by first building the parse tree of the problem based on the acquired grammar, and then extracting subsequences associated with feature symbols from the parse tree as target features. This model presented so far learns to extract deep features in a mostly unsupervised way without any goals or context from SimStudent problem solving. Later, we describe how to extend its ability by integrating it into SimStudent.

2.3 Empirical Evaluation on Deep Feature Learner

To evaluate the proposed deep feature learner, we carried out two controlled experiments. We compared four alternatives of the proposed approach: 1) without transfer learning and no feature focus; 2) without transfer learning, but with feature focus; 3) with transfer learning (from unlabeled sub-grammars), and without feature focus; 4) with transfer learning from unlabeled sub-grammars and feature focus. Learners without labeled feature have no way of knowing what the feature is; instead, we report the accuracy that would be obtained using the non-terminal symbol that mostly frequently corresponds to the feature sub-grammar in the training examples. Note that we did not compare the proposed deep feature learner with the inside-outside algorithm, as Li et al. have shown that the base learner (i.e. the learner with no extension) outperforms the inside-outside algorithm.[1] All the experiments were run on a 2.53 GHz Core 2 Duo Mackbook with 4GB of RAM.

In order to understand the generality and scalability of the proposed approach, we first designed and carried out experiments in synthetic domains. The experiment results show that the learner with both transfer learning and feature focus (*+Transfer +Feature Focus*) has the steepest learning curve. Learners with a single extension (*-Transfer +Feature Focus*,

[1] http://rakaposhi.eas.asu.edu/nan-tist.pdf

Table 2.2: Method summary

Three tasks:	T1, learn signed number
	T2, learn to find coefficient from expression
	T3, learn to find constant from equation
Three curricula:	T1 → T2
	T2 → T3
	T1 → T2 → T3
Number of training condition:	10
Training size in all but last tasks:	10
Training size in the last task:	1, 2, 3, 4, 5
Testing size:	100

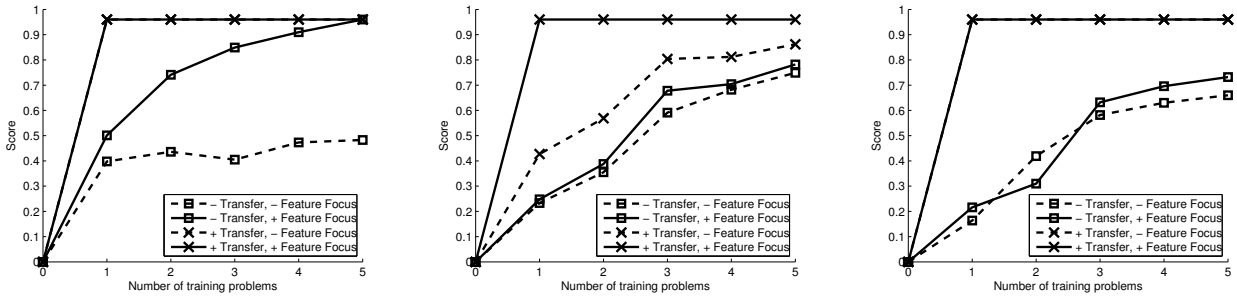
and *+Transfer -Feature Focus*) have a slower learning curve comparing with the learner with both extensions (*+Transfer +Feature Focus*), but both outperform the base learner (*-Transfer -Feature Focus*). All learners acquire the targeted feature within a reasonable amount of time, i.e., 1 – 266 millliseconds per training record with domains of size smaller than or equals to 25. For full details, pleaser refer to [54].

In order to understand whether the proposed algorithm is a good model of real students, we carried out a controlled simulation study in algebra. Accelerated future learning, in which learning proceeds more effectively and more rapidly because of prior learning, is an interesting measure of robust learning. Two possible causes for accelerated future learning are a better learning strategy or stronger prior knowledge. Learning with feature focus is an example of using a better learning strategy during knowledge acquisition. Transfer learning from unlabeled sub-grammars is an example of developing stronger prior knowledge from previous training to prepare for better future learning. The objective of this study is to test 1) whether the proposed model could yield accelerated future learning with stronger prior knowledge and better learning strategies, 2) if so, how prior knowledge and learning strategies affect the learning outcome. In other words, can we model how students may learn later tasks more effectively after prior unsupervised or semi-supervised experience?

2.3.1 Method

In order to understand the behavior of the proposed model, we designed three curricula. Three tasks are used across the three curricula. Task one is to learn about signed numbers. Task two is to learn how to recognize a coefficient from expressions in the form of *{SignedNumber x}*. Task three is to learn how to recognize a constant in the left-hand side from equations in the form of *{SignedNumber x - Integer = SignedNumber}*. The three curricula contain 1) task one, task two; 2) task two, task three; 3) task one, task two, and task three.

There were also 10 training sequences to control for a difference in training problems. The training data were randomly generated following the grammar corresponding to each task. For instance, task two's grammar is shown in Table 2.1. In all but the last task, each learner was given 10 training problems. For the last task, each learner was given one to five training records.



(a) (b) (c)

Figure 2.6: Learning curves in the last task for four learners in curriculum (a) from task one to task two (b) from task two to task three (c) from task one and two to task three. Both prior knowledge transfer and the feature focus strategy produce faster learning.

To measure learning gain under each training condition both systems were tested on 100 expressions in the same form of the training data in the last task. For each testing record, we compared the feature extracted by the oracle grammars with that recognized by the acquired grammars. Note that in task two, 4% of the testing problems in task two were x and $-x$. To assess the accuracy of the model, we asked both systems to extract the feature from each problem. We then used the oracle grammar to evaluate the correctness of output. A brief summary of the method is shown in Table 2.2.

2.3.2 Measurements

To assess the learning outcome, we measured the learning rate in the last task of each curriculum to evaluate the effectiveness of the learners. The experiment tested whether the proposed model is able to yield accelerated future learning, that is, a faster learning rate in the last task either because of transfer prior learning or because of a better learning strategy. We compare the same four learners used in the previous simulations, that is, the combinations of transfer or not and feature focus or not. To evaluate the learning rate, we report learning curves for all four learners by the number of training problems given in the last task. The accuracy of the feature extraction task is averaged over 10 training conditions.

This experiment focuses on measuring the learning rate. In section 2.6, we also test whether the proposed model fits with real student data. We show that after integrating the proposed model into a simulated student, the extended simulated student can be used to automatically discover student models. The discovered model fits with real student data better than human-generated models. This indicates that the extended simulated student simulates the real student learning process well.

2.3.3 Evaluating Accelerated Future Learning

As shown in Figure 6(a), with curriculum one, all four learners acquired better knowledge with more training examples. With five training problems, either transfer or feature focus are sufficient to acquire knowledge of score 0.96, while the base learner with neither was

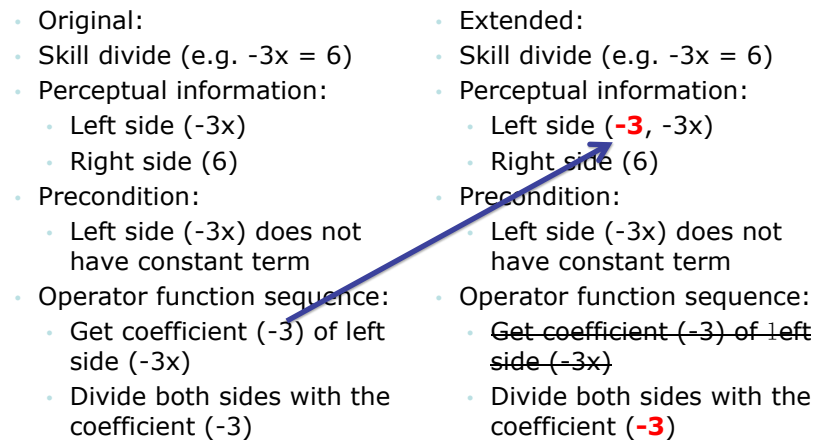


Figure 2.7: Original and extended production rules for divide in a readable format. Grammar learning allows extraction of information in where-part of the production rule and eliminated the need for domain-specific function authoring (get-coefficient) for use in the how-part.

only able to achieve a score around 0.5. Both learners with transfer learning (*+Transfer -Feature Focus*, and *+Transfer +Feature Focus*) have the steepest learning curve. In fact, they reached a score of 0.96 with only one training example. The feature focus learner (*-Transfer +Feature Focus*) learns more slowly than the learners with transfer learning (*+Transfer -Feature Focus*, and *+Transfer +Feature Focus*), but is able to reach a score of 0.96 after five training examples. Learners that transfer prior grammar learning achieve faster future learning than those without transfer learning. The base learner (*-Transfer -Feature Focus*) learns most slowly. A careful inspection shows that without feature focus and transfer learning, the base learner was not able to acquire a grammar rule with a non-terminal symbol generally corresponding with the feature "coefficient", though it does learn to identify positive coefficients (like many novice students). This causes the failure of identifying the feature symbol. Comparing the base learner (*-Transfer -Feature Focus*) and the learner with feature focus (*-Transfer +Feature Focus*) we can see that a better learning strategy also yields a steeper learning curve.

Similar results were also observed with curriculum two and curriculum three. In curriculum two, one interesting point is that, in some conditions, if a transfer learner, (*+Transfer -Feature Focus*) remembers the wrong knowledge acquired from task two, and transferred this knowledge to task three, the learner will perform even worse than the learner with no prior knowledge (*-Transfer -Feature Focus*). This indicates that more knowledge does not necessarily lead to steeper learning curves. Transferring incorrect knowledge leads to less learning.

In all three curricula, the transfer learner (*+Transfer -Feature Focus*) always outperforms the learner with the semantic non-terminal constraint (*-Transfer +Feature Focus*). This suggests that prior knowledge is more effective in accelerating future learning than this learning strategy.

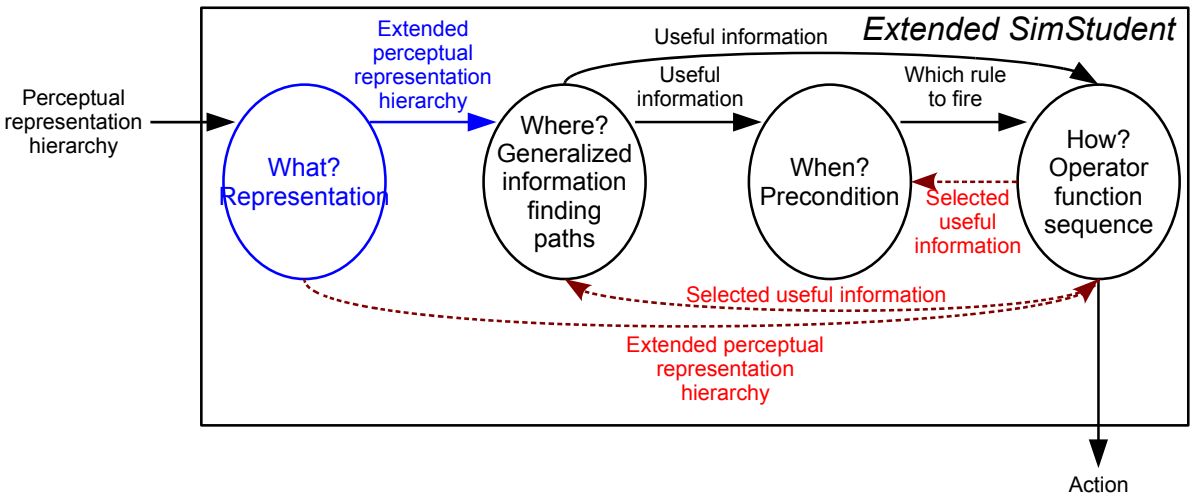


Figure 2.8: A diagram about how the extended SimStudent makes use of the representation acquired by the deep feature learning, and reacts with the environment. Solid lines show the information that flows through components during execution. Red dashed lines illustrates how the acquired representation is used by the learning components during learning. Information about the training examples is not presented.

2.4 Integrating Deep Feature Representation Learning into SimStudent

Given the promising results shown above, we believe the proposed deep feature learner is effective in acquiring representation knowledge, and is a good model of real students. To evaluate how the deep feature representation learner could affect the problem-solving learning of an intelligent agent, in this section, we present an integration of deep feature learning into such an agent, SimStudent. As we have mentioned above, SimStudent is able to acquire production rules in solving complicated problems, but requires a set of operator functions given as prior knowledge. Some of the operator functions are domain-specific and require expert knowledge to build. In contrast, the feature learner acquires deep features that are essential for effective learning without requiring prior knowledge engineering. In order to both reduce the amount of prior knowledge engineering needed for SimStudent and to build a better model of real students, we present a novel approach that integrates the representation learner into SimStudent. Figure 2.7 shows a comparison between a production rule acquired by the original SimStudent and the corresponding production rule acquired by the extended SimStudent. As we can see, the coefficient of the left-hand side (i.e., *-3*) is included in the perceptual information part in the extended production rule. Therefore, the operator function sequence no longer needs the domain-specific operator, "get-coefficient". To achieve this, we extended the perceptual learning algorithm as described below.

Figure 2.8 shows a high-level diagram that illustrates how the extended percept hierarchy is used by the learning components with red dashed lines. We first extend the perceptual representation hierarchy by the representation acquired by the deep feature learner, and then sends the extended hierarchy to the how learner. The how learner finds an operator function sequence with the extended hierarchy, and selects a subset of the elements in the extended hierarchy. The where and when learners then carry out their learning processes with this

selected set of useful information.

To improve perceptual representation, we extend the percept hierarchy of GUI elements to further include the most probable parse tree for the content in the leaf nodes (e.g., text fields) by appending the parse trees as an extension of the GUI path learning to the associated leaf nodes. All of the inserted nodes are of type "subcell". In the algebra example, this extension means that for cells that represent expressions corresponding to the sides of the equation, the extended SimStudent appends the parse trees for these expressions to the cell nodes. Let's use *-3x* as an example. In this case, the extended hierarchy includes the parse tree for *-3x* as shown at the left side of Figure 2.5 as a subtree connecting to the cell node associated with *-3x*. With this extension, the coefficient (*-3*) of *-3x* is now explicitly represented in the percept hierarchy. If the extended SimStudent includes this subcell as a percept in production rules, as shown at the right side of Figure 2.7, the new production rule does not need the first domain-specific operator function "coefficient".

However, extending the percept hierarchy presents challenges to the original perceptual learner. First of all, since the extended subcells are not associated with GUI elements, we can no longer depend on the author to specify relevant perceptual input for SimStudent, nor can we simply specify all of the subcells in the parse trees as relevant perceptual information; otherwise, the acquired production rules would include redundant information that would hurt the generalization capability of the perceptual learner. Second, since the size of the parse tree for an input depends on the input length, the fixed percept size assumption made by SimStudent no longer holds. Even with the same number of percepts, how the inserted percepts should be ordered is not immediately clear. To address these challenges, we extend the original perceptual learner to support acquisition of perceptual information with redundant and variable-length percept lists.

To do this, SimStudent first includes all of the inserted subcells as candidate percepts, and calls the operator function sequence learner to find an operator function sequence that explains all of the training examples. In our example, the operator function sequence for *(divide -3)* would only contain one operator function "divide", since *-3* is already included in the candidate percept list. The perceptual learner then removes all of the subcells that are not used by the operator function sequence from the candidate percept list. Hence, subcells such as -, *3* and *x* would not be included in the percept list any more. Since all of the training example action records share the same operator function sequence, the number of percepts remaining for each example action record should be the same. Next, the percept learner arranges the remaining subcell percepts based on their order of being used by the operator function sequences. After this process, the percept learner now has a set of percept lists that contains a fixed number of percepts ordered in the same fashion. We can then switch to the original percept learner to find the least general paths for the updated percept lists. In our example for skill "divide", as shown at the right side of Figure 2.7, the perceptual information part of the production rule would contain three elements, the left-hand side and right-hand side cells which are the same as the original rule, and a coefficient subcell which corresponds to the left child of the variable term. Note that since we removed the redundant subcells, the acquired production rule now works with both *-3x=6* and *4x=8*.

	Operator functions Userd	
	Strong Domain-Specific	Weak Domain-General
Original+Strong Ops	8	7.5
Extended+Weak Ops	0	12
Original+Weak Ops	0	14.5

Table 2.3: Average number of strong and weak operator functions used in acquired production rules.

2.5 Experimental Study on SimStudent Integrated with Deep Feature Learner

In order to evaluate whether the extended SimStudent is able to acquire correct knowledge with reduced prior knowledge engineering, we carried out an experiment in the algebra domain. We use algebra as the testing domain because it is one of the most important learning tasks for middle school students. It is also relatively more complicated than other similar domains such as multi-column addition and fraction addition. Although not reported here, we have demonstrated elsewhere [53] that the extended SimStudent yields better learning performance, with less knowledge engineering, than the original SimStudent in fraction addition and stoichiometry.

2.5.1 Experiment Design

Since our goal is to build an intelligent agent that models skill acquisition of real students, instead of using randomly generated problems, as training sets four problem sets we select that were used to teach real students as training sets. More specifically, the problem sets are from high school students who used Carnegie Learning Algebra I Tutor. The sizes of the training sets are 13, 14, 35 and 35 problems. We also choose 10 other problems from real student data as the testing set.

We compare the extended SimStudent with the original SimStudent given different amounts of prior knowledge. The extended SimStudent is first trained on a sequence of deep feature learning tasks, which include learning what is a signed number, what is a term, and what is an expression. We then construct a weak operator function set and a strong operator function set, simulating weak and strong prior knowledge. The weak operator function set contains 24 domain-general operator functions such as copying a string, adding two numbers and so on. The strong operator function set includes the weak operator function set plus 12 domain-specific operator functions such as getting the coefficient, adding two terms and so on. The list of operator functions that are used in the production rules acquired by the four SimStudents can be found in [54]. Two original SimStudents and one extended SimStudent are tested. One of the original SimStudents is given the strong operator function set (*O+Strong Ops*), while the other is provided with the weak operator function set (*O+Weak Ops*). The extended SimStudent is given only the weak operator function set (*E+Weak Ops*).

2.5.2 Experiment Results

Evaluation of Learning Speed: The first study we carried out focuses on evaluation of learning speed. Since it is often possible to have more than one way of solving the same

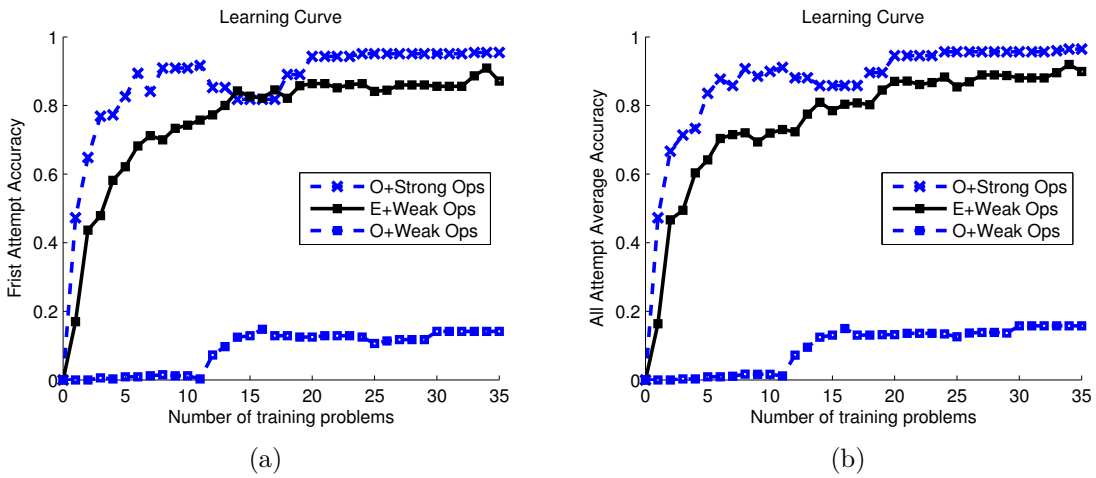


Figure 2.9: Learning curves for three learners (a) first-attempt accuracy (b) all-attempt average accuracy.

algebra equation, it is also possible that there is more than one skill applicable at the same time. In order to evaluate the performance of all applicable skills, we use two different measurements in evaluating the learning efficiency. The first measurement is called *first-attempt accuracy*, where for each testing problem, the learner receives score 1 if it proposes a correct step at its first attempt, and gets 0 otherwise. This measurement is closest to the evaluation method used in real classroom settings, where even if the student has more than one thought in solving the problem, only the one solution he/she writes out is graded. The second measurement, *all-attempt average accuracy*, focuses more on the average performance across all applicable skills. Instead of only counting for the first attempt, the evaluator scores the correctness of all applicable skills, and reports the average score as the all-attempt average accuracy.

The average learning curves for the three SimStudents are shown in Figures 9(a) and 9(b). The blue lines correspond to the original SimStudents, whereas the black lines represent the performance of the extended SimStudent. As we can see in the figures, with both measurements, there is a huge gap between the two original SimStudents with (*O+Strong Ops*) and without (*O+Weak Ops*) strong operator functions. Our focus is to test whether the extended SimStudent is able to achieve performance comparable to the original SimStudent with strong operator functions (*O+Strong Ops*) while given only the weak operator function set. As the result shows, the extended SimStudent (*E+Weak Ops*) learns slower than the original SimStudent with strong operator functions (*O+Strong Ops*) at the very beginning, but gradually catches up with the original SimStudent. With 35 training problems, the all-attempt average accuracy of the extended SimStudent (*E+Weak Ops*) reaches 90%, which is only 5% lower than the original SimStudent with strong operator functions (*O+Strong Ops*). This suggests that with the deep feature learner, the extended SimStudent is able to achieve comparable performance without prior domain-specific knowledge engineering.

Evaluation of Knowledge Engineering Needed: We evaluate the learner performance with two measurements, the total amount of knowledge used and the learning speed. For the first measurement, we look at the production rules acquired from the two problem sets of size 35, and report the average number of domain-specific and domain-general operator

functions used in the two rule sets. Recall that domain-specific operator functions usually require more knowledge engineering than domain-general operator functions. As shown in Table 2.3, the SimStudent (*O+Strong Ops*) that was given strong operator functions used 8 of the domain-specific operator functions, plus 7.5 domain-general operator functions on average across the two training sequences of size 35. In contrast, the extended SimStudent (*E+Weak Ops*) was not given any domain-specific functions and used 12 domain-general operator functions, which indicates much less knowledge engineering effort. In addition, the original SimStudent with only domain-general operator functions (*O+Weak Ops*) used 14.5 domain-general operator functions, which suggests that it needs a larger amount of prior knowledge engineering than the extended SimStudents. However, as we have seen before, it performs much worse than the extended SimStudents.

2.6 Using SimStudent to Discover Better Learner Models

As mentioned above, we are not only interested in building a learning agent: we would also like to construct a learning agent that simulates how students acquire knowledge. In this section, we are going to present an approach that automatically discovers learner models using the extended SimStudent. If the discovered model turns out to be a good learner model, we should be able to conclude that the extended SimStudent simulates the real student learning process well. A learner model is a set of *knowledge components (KC)* encoded in intelligent tutors to model how students solve problems. The set of KCs includes the component skills, concepts, or percepts that a student must acquire to be successful on the target tasks. For example, a KC in algebra can be how students should proceed given problems of the form *Nv=N* (e.g., *-3x = 6*). The learner model provides important information to automated tutoring systems in making instructional decisions. Better learner models match with real student learning behavior, that is, changes in performance over time. They are capable of predicting task difficulty and transfer of learning between related problems, and can be used to yield better instruction.

Traditional ways to construct models include structured interviews, think-aloud protocols, rational analysis, and so on. However, these methods are often time-consuming, and require expert input. More importantly, they are highly subjective. Previous studies [39, 40] have shown that human engineering of these models often ignores distinctions in content and learning that have important instructional implications. Other methods such as Learning Factor Analysis (LFA) [14] apply an automated search technique to discover learner models. It has been shown that these automated methods are able to find better learner models than human-generated ones. Nevertheless, LFA requires a set of human-provided factors given as input. These factors are potential KCs. LFA carries out the search process only within the space of such factors. If a better model exists but requires unknown factors, LFA will not find it.

To address this issue, we propose a method that automatically discovers learner models without depending on human-provided factors. The system uses the extended SimStudent to acquire skill knowledge. Each production rule corresponds to a KC that students need to learn. The model then labels each observation of a real student based on skill application.

2.6.1 Method

In order to evaluate the effectiveness of the proposed approach, we carried out a study using an algebra dataset. We compared the SimStudent model with a human-generated KC model by first coding the real student steps using the two models, and then testing how well the two model codings predict real student data. Note that DataShop [37] has 21 different KC models for the current study, the human-generated KC model we selected here is one of the best models among the existing student models.

For the human-generated model, the real student steps were first coded using the "action" label associated with a correct step transaction, where an action corresponds to a mathematical operation(s) to transform an equation into another in a way that makes progress toward the solution. As a result, there were nine KCs defined (called the Action KC model) – add, subtract, multiply, divide, distribute, clt (combine like terms), mt (simplify multiplication), and rf (reduce a fraction). Four KCs associated with the basic arithmetic operations (i.e., add, subtract, multiply, and divide) were then further split into two KCs for each, namely a skill to identify an appropriate basic operator and a skill to actually execute the basic operator. The former is called a *transformation* skill whereas the latter is a *typein* skill. As a consequence, there were 12 KCs defined (called the Action-Typein KC model). Not all steps in the algebra dataset were coded with these KC models – some steps are about a transformation that we do not include in the Action KC model (e.g., simplify division). There were 9487 steps that can be coded by both KC models mentioned above. The "default" KC model, which were defined by the productions implemented for the cognitive tutor, has only 6809 steps that can be coded. To make a fair comparison between the "default" and "Action- Typein" KC models, we took the intersection of those 9487 and 6809 steps. As a result, there were 6507 steps that can be coded by both the default and the Action-Typein KC models. We then defined a new KC model, called the Balanced-Action-Typein KC model that has the same set of KCs as the Action-Typein model but is only associated with these 6507 steps, and used this KC model to compare with the SimStudent model.

To generate the SimStudent model, SimStudent was tutored on how to solve linear equations by interacting with the Carnegie Learning Algebra I Tutor, like a human student. As the training set for SimStudent, we selected 40 problems that were used to teach real students. Given all of the acquired production rules, for each step a real student performed, we assigned the applicable production rule as the KC associated with that step. In cases where there was no applicable production rule, we coded the step using the human-generated KC model (Balanced-Action-Typein). Each time a student encounters a step using some KC, it is considered as an "opportunity" for that student to show mastery of that KC, and learn the KC by practicing it.

Having finished coding real student steps with both models (the SimStudent model and the human-generated model), we used the Additive Factor Model (AFM) [14] to validate the coded steps. AFM is an instance of logistic regression that models student success using each student, each KC, and the KC by opportunity interaction as independent variables,

$$\ln \frac{p_{ij}}{1 - p_{ij}} = \theta_i + \sum_k \beta_k Q_{kj} + \sum_k Q_{kj}(\gamma_k N_{ik}) \tag{2.1}$$

Where:

i represents a student i.

j represents a step j.

k represents a skill or KC k.

p_{ij} is the probability that student i would be correct on step j.

θ_i is the coefficient for proficiency of student i.

β_k is coefficient for difficulty of the skill or KC k

Q_{kj} is the Q-matrix cell for step j using skill k.

γ_k is the coefficient for the learning rate of skill k;

N_{ik} is the number of practice opportunities student i has had on the skill k;

We utilized DataShop [37], a large repository that contains datasets from various educational domains as well as a set of associated visualization and analysis tools, to facilitate the process of evaluation, which includes generating learning curve visualization, AFM parameter estimation, and evaluation statistics including AIC (Akaike Information Criterion) and cross validation.

2.6.2 Dataset

We analyzed the same data from 71 students who used an Carnegie Learning Algebra I Tutor unit on equation solving. The students were typical students at a vocational-technical school in a rural/suburban area outside of Pittsburgh, PA. The problems varied in complexity, for example, from simpler problems like *3x=6* to harder problems like *x/-5+7=2*. A total of 19,683 transactions between the students and the Algebra Tutor were recorded, where each transaction represents an attempt or inquiry made by the student, and the feedback given by the tutor.

2.6.3 Measurements

To test how well the existing and generated models predict with real student data, we used AIC and a 3-fold cross validation. AIC measures the fit to student data while penalizing over-fitting. The cross validation was performed over three folds with the constraint that each of the three training sets must have data points for each student and KC. We report the root mean-squared error (RMSE) averaged over three test sets.

2.6.4 Experiment Results

The SimStudent model contains 21 KCs. Both the AIC (6448) and the cross validation RMSE (0.3997) are lower than the human-generated model (AIC 6529 and cross validation 0.4034). This indicates that the SimStudent model better predicts real student behavior.

In order to understand whether the differences are statistically reliable or not, we carried out two significance tests. The first significance test evaluates whether the SimStudent model

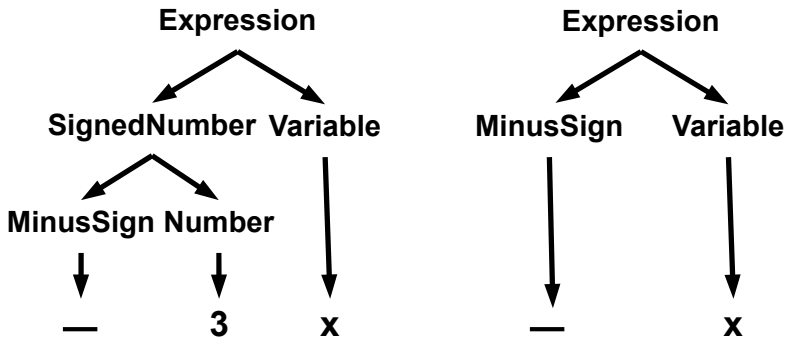


Figure 2.10: Different parse trees for -3x and -x.

is actually able to make better predictions than the human-generated model. During the cross validation process, each student step was used once as the test problem. We took the predicated error rates generated by the two KC models for each step during testing. Then, we compared the KC models' predictions with the real student error rate (0 if the student was correct at the first attempt, and 1 otherwise). After removing ties, among all 6494 student steps, the SimStudent model made better predictions than the human-generated KC model on 4260 steps. A sign test on this shows that the SimStudent model is significantly ($p < 0.001$) better in predicting real student behavior than the human-generated model. In the second test, due to the random nature of the assignment to folds in cross validation, we evaluated whether the lower RMSE achieved by the SimStudent model was consistent or could be due to chance. To do this, we repeated the cross validation 20 times, and calculated the RMSE for both models. Across the 20 runs, the SimStudent model consistently outperformed the human-generated model: in particular, a paired t-test shows the SimStudent model is significantly ($p < 0.001$) better than the human-generated model. [2] Therefore, we conclude that the SimStudent model is a reliably better student model than the human-generated KC model.

2.6.5 Implications for Instructional Decision

We can inspect the data more closely to get a better qualitative understanding of why the SimStudent model is better and what implications there might be for improved instruction. Among the 21 KCs learned by the SimStudent model, there were 17 transformation KCs and four typein KCs. It is hard to map the SimStudent KC model directly to the expert model. Approximately speaking, the distribute, clt (i.e. combine like terms), mt, rf KCs as well as the four typein KCs are similar to the KCs defined in the expert model. The transformation skills associated with the basic arithmetic operators (i.e., add, subtract, multiply and divide) are further split into finer grain sizes based on different problem forms.

One example of such split is that SimStudent created two KCs for division. The first KC

[2] Note that differences between competitors in the KDD Cup 2010 (https://pslcdatashop.web.cmu.edu/KDDCup/Leaderboard) have also been in this range of thousands in RMSE.

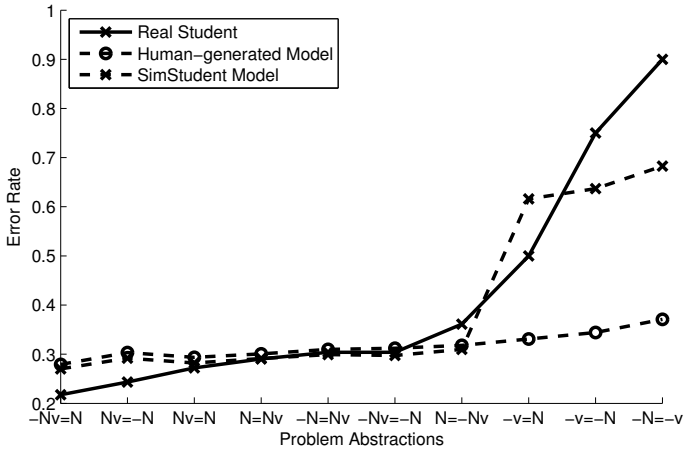


Figure 2.11: Error rates for real students and predicted error rates from two student models.

(simSt-divide) corresponds to problems of the form $Ax=B$, where both A and B are signed numbers, whereas the second KC (simSt-divide-1) is specifically associated with problems of the form $-x=A$, where A is a signed number. This is caused by the different parse trees for Ax vs $-x$ as shown in Figure 2.10. To solve $Ax=B$, SimStudent simply needs to divide both sides with the signed number A. On the other hand, since $-x$ does not have -1 represented explicitly in the parse tree, SimStudent needs to see $-x$ as $-1x$, and then to extract -1 as the coefficient. If SimStudent is a good model of human learning, we expect the same to be true for human students. That is, real students should have greater difficulty in making the correct move on steps like $-x = 6$ than on steps like $-3x = 6$ because of the need to convert (perhaps just mentally) $-x$ to $-1x$. To evaluate this hypothesis, we computed the average error rates for a relevant set of problem types – these are shown with the solid line in Figure 2.11 with the problem types defined in forms like $-Nv=N$, where the Ns are any integrate number and the v is a variable (e.g., $-3x=6$ is an instance of $-Nv=N$ and $-6=-x$ is an instance of $-N=-v$). The problem types are sorted by increasing error rates. In other words, the problem types to the right are harder for human students than those to the left.

We also calculated the mean of the predicted error rates for each problem type for both the human-generated model and the SimStudent model. Consistent with the hypothesis, as shown in Figure 2.11, we see that problems of the form $Ax=B$ (average error rate 0.283) are much simpler than problems of the form $-x=A$ (average error rate 0.719). The human-generated model predicts all problem types with similar error rates (average predicted error rate for $Ax=B$ 0.302, average predicted error rate for $-x=A$ 0.334), and thus fails to capture the difficulty difference between the two problem types ($Ax=B$ and $-x=A$). The SimStudent model, on the other hand, fits with the real student error rates much better. It predicts higher error rates (0.633 on average) for problems of the form $-x=A$ than problems of the form $Ax=B$ (0.291 on average).

SimStudent's split of the original division KC into two KCs, simSt-divide and simSt-divide-1, suggests that the tutor should teach real students to solve two types of division problems separately. In other words, when tutoring students with division problems, we should include two subsets of problems, one subset corresponding to simSt-divide problems

(Ax=B), and one specifically for simSt-divide-1 problems (-x=A). We should perhaps also include explicit instruction that highlights for students that -x is the same as -*1x*.

2.7 Related Work

The main contribution of this thesis is to build a human-like intelligent agent, and to reduce the amount of knowledge engineering required, by integrating representation learning into an agent. We exploit the connection between representation learning and grammar induction by extending an existing pCFG algorithm [56] to support feature learning and transfer learning. It shares some ideas from previous work on grammar induction (e.g., [46, 90, 97, 100]), which searches for the target grammar by adding or merging non-terminal symbols. Roark and Bacchiani [81], Hwa [32], and others have also explored transfer learning for pCFGs. But most of the above approaches focus on the grammar induction task, rather than applying the techniques to representation learning as we do here.

Previous work in cognitive science has shown that "chunking" is an important component of human knowledge acquisition. Theories of the chunking mechanisms [16, 28, 79] have been constructed. EPAM [16] is one of the first chunking theories proposed to explain key phenomena of expertise in chess. Learning occurs through the incremental growth of a discrimination network, where each node in the network is a chunk. It has been shown that chunks can be used to suggest plans, moves and so on. A later version of EPAM, EPAM-IV [79], extends the basic chunking mechanism to support a retrieval structure that enables domain-specific material to be rapidly indexed. In these theories, chunks usually refer to perceptual chunks. In addition, CHREST [28] proposes a template theory, where the discrimination network contains both perceptual chunks and action chunks. A more detailed review of these work can be found in [27]. Our work is similar to these work as we are also modeling the learning of perceptual chunks, a kind of deep feature learning, but differs from these theories since none of the above theories uses pCFG learning to model the acquisition of perceptual chunks.

There has also been considerable research on learning within agent architectures. Soar [42] uses a chunking mechanism to acquire knowledge that constrains problem-space search. Another architecture ACT-R [1] creates new production rules through a compilation process that gradually transforms declarative representations into skill knowledge [91]. Anderson and Thompson [2] developed an analogical problem solving mechanism, and integrated it into an earlier version of ACT-R to assist skill learning. Icarus [44] acquires complex hierarchical skills in the context of problem solving. Unlike those theories, SimStudent puts more emphasis on knowledge-level learning (cf., [69]) achieved through induction from positive and negative examples. It integrates ideas of theories of perceptual chunking [79] as a basis for improving knowledge representations that, in turn, facilitate better learning of problem solving skills.

Another closely related research area is learning procedural knowledge by observing others' behavior. Classical approaches include explanation-based learning [65, 83], learning apprentices [64] and programming by demonstration [20, 48]. Neves [68] proposed a program that learns production rules from worked-out solutions and by working problems, and demonstrated the algorithm in algebra, but did not show results across domains. Most of these approaches used analytic methods to acquire candidate procedures. Other works on transfer

learning (e.g., [70, 76, 78, 94]) also share some resemblance with our work. They focus on improving the performance of learning by transferring previously acquired knowledge from another domain of interest. However, to the best of our knowledge, none of the above approaches uses the transfer learner to acquire a better representation that reveals essential percept features, and to integrate it into an intelligent agent.

Other research in cognitive science also attempts to use probabilistic approaches to model the process of human learning. Kemp and Xu [35] applie a probabilistic model to capture principles of infant object perception. Kemp and Tenenbaum [34] used a hierarchical generative model to show the acquisition process of domain-specific structural constraints. But again, neither of the above approaches tend to use the probabilistic model as a representation acquisition component in a learning agent. Additionally, research on deep architectures [7] shares a clear resemblance with our work and has been receiving increasing attention recently. Theoretical results suggest that in order to learn complicated functions such as AI-level tasks, deep architectures that are composed of multiple levels of non-linear operation are needed. Although not having been studied much in the machine learning literature due to the difficulty in optimization, there are some notable exceptions in the area including convolutional neural networks [49, 50, 77, 86], sigmoidal belief networks learned using variational approximations [21, 30, 82, 93], and deep belief networks [8, 29]. While both the work in deep architectures and our work are interested in modeling complicated functions through non-linear features, the tasks we work on are different. Deep architectures are used more often in classification tasks whereas our work focuses on simulating human problem solving and learning of math and science.

In this work, it is demonstrated that we can use SimStudent to build better cognitive models of student performance. A lot of efforts have also been put toward comparing the quality of alternative student models. LFA automatically discovers student models, but is limited to the space of the human-provided factors. Other works such as [72, 99] are less dependent on human labeling, but may suffer from challenges in interpreting the results. In contrast, the SimStudent approach has the benefit that the acquired production rules have a precise and usually straightforward interpretation. Other systems [6, 92] use a Q-matrix to find knowledge structure from student response data. Baffes and Mooney [5] apply theory refinement to the problem of modeling incorrect student behavior. Langley and Ohlsson's [45] ACM applies symbolic machine learning techniques to automatically construct student models. Brown and Burton's [11] DEBUGGY, and Sleeman and Smith's [88] LMS also make use of artificial intelligent tools to construct models that explain student's behavior in math domains. VanLehn's [96] Sierra models the impasse-driven acquisition of hierarchical procedures for multi-column subtraction from sample solutions. However, his work focused on explaining the origin of bugs for real students, which is not the focus here. In addition, Sierra is given a CFG for parsing the visual state of the subtraction problems, whereas our system automatically acquires a pCFG. Besides SimStudent, there has been a lot of work on creating simulated students [15, 73]. None of the above approaches focused on modeling how representation learning affects skill learning. Moreover, none of them compared the system with human learning curve data. To the best of our knowledge, our work is the first combination of the two whereby we use student model evaluation techniques to assess the quality of a simulated learner.

Ohlsson [71] reviews how different learning models are employed during different learning

phases in intelligent systems. Our work on integrating representation learning and skill learning also reflects how one learning mechanism is able to aid other learning processes in an intelligent systems.

3 Proposed Work

For the rest of this work, we would like to further explore how representation learning can affect the performance of skill learning. We will explore this topic in four directions.

3.1 Learning Other Kinds of Perceptual Knowledge

In previous work, by extending the percept hierarchy with the representation acquired by the deep feature leaner, we have successfully demonstrated that SimStudent is able to learn skill knowledge effectively using only domain-general operator functions. However, this is not the only type of perceptual knowledge learned by human students. Both the hierarchy of the elements in the interface to support the where-part learning, and the set of feature predicates to the when-part learning are representation knowledge that is key to the success of learning. In the rest of this section, we would like to further extend the deep feature learner to accommodate acquisition of other kinds of perceptual knowledge. By doing this, we expect to further increase the general interlectual flexibility, and thereby reduce the amount of knowledge engineering required in building an intelligent agent.

3.1.1 Learning to Perceive Two-Dimensional Displays Using Probabilistic Grammars

Every day, people view and understand many novel two-dimensional (2-D) displays such as tables on webpages and software user interfaces. How do humans learn to process such displays? As an example, Figure 2.1 shows a screenshot of one interface to an intelligent tutoring system that is used to teach students how to solve algebraic equations. The interface should be viewed as a table of three columns, where the first two columns of each row contain the left-hand side and right-hand side of the equation, and the third column names the skill applied. In tutoring, students enter data row by row, a strategy which requires a correct intuitive understanding of how the interface is organized. To learn effectively, SimStudent needs a similar understanding of the way the interface is organized. Past instances of SimStudent have used a hand-coded hierarchical representation of the interface; here we consider replacing that hand-coded element with a learned representation.

More generally, we consider using a two-dimensional (2-D) variant of a probabilistic context-free grammar (pCFG) to model how a user perceives the structure of a user interface. Hence, we need to extend the 1-D pCFG learner to support acquisition of 2-D pCFGs. One of the major challenges in developing this algorithm would be that unlike the 1-D case, it is not clear whether we should merge elements in the interface horizontally or vertically, or sometimes horizontally and sometimes vertically. As shown in Figure 2.1, even if $\langle \mathit{Cell11}, \mathit{Cell21} \rangle$, $\langle \mathit{Cell11}, \mathit{Cell12} \rangle$ are both adjacent pairs of algebraic expressions. Merging *Cell 11* and *Cell 21* would be less plausible as $\langle \mathit{Cell11}, \mathit{Cell12} \rangle$ forms an equation, whereas the other pair does not. Nevertheless, such information is not obtainable based on the spatial layout of the interface, nor can it be known ahead of time by the novice algebra

learner. In response, we plan to exploit temporal information about when users interact with the interface in addition to the spatial layout of the interface. The alphabet of the grammar is a vocabulary of non-terminal symbols representing primitive interface-element types. For example, in Figure 2.1, the type of the cells in the first two columns is *Expression*, and the type of the last cell in the each column is *Skill*. (These primitive types can be learned from prior experience by the 1-D deep feature learner.)

We will then integrate this two-dimensional representation learner into SimStudent by replacing the hand-coded display representation with the statistically learned display representation. We plan to demonstrate the proposed algorithm in multiple skill acquisition domains. In order to assess whether the proposed learning algorithm is effective in modeling unsupervised perceptual learning of layout representation, we would like to evaluate the proposed algorithms in both synthetic domains and real world domains without integration into SimStudent. We then would like to test how the acquired representation affects SimStudent's learning effectiveness in comparison with a manually created representation in three domains.

Related Work: Two-dimensional pCFGs have been used in other domains such as recognizing equations (e.g., [18,97]) and classifying images (e.g., [87]). Algorithms in this direction often assume the structure of the grammar or the parse structures of the training examples is given, and use a two-dimensional parsing algorithm to find the most likely parse of the observed image. We would like to also model the learning process of the structure of the grammar, and apply the technique to another domain, learning to perceive user interface.

Research on extracting structured data on the web (e.g., [4, 12, 19, 24, 58, 62, 104]) shares a clear resemblance with the proposed approach, as it also concerns on understanding structures embedded in a two-dimensional space. It differs from our work in that webpages have an observable hierarchical structure in the form of their HTML parse trees, whereas we only observe the 2-D visual displays, which have no such structural information.

3.1.2 Creating Features from a Learned Grammar

In past work, the acquired "deep" features could be exploited only in learning the ***where*** and ***how*** parts of a skill, and have been shown to generate as good or better performance while requiring much less knowledge engineering. In this work, we consider also using deep features to learn the ***when*** part of a skill. As suggested by its name, the deep feature learner acquires predicate representations that reveal deep functional features that are essential for effective future learning. Hence, instead of manually constructing a set of feature predicates, we plan to extend the deep feature learner to automatically generate a set of feature predicates based on the parse trees of the input strings (e.g., whether a term has a signed number in it), and provide these automatically generated feature predicates as prior knowledge for SimStudent.

More specifically, we propose to automatically generate, from the acquired "deep" features, a set of predicates that can be used by the inductive logic programming (ILP) component that learns when to apply a skill. These automatically generated feature predicates can then replace manually constructed feature predicates. Then the question comes which feature predicates should we generate. On the one hand, the set of automatically generated feature predicates should be rich enough so that it provides enough description for FOIL to

differentiate positive examples from negative examples. On the other hand, we should not generate too many feature predicates, since it will increase the learning complexity of FOIL.

We plan to generate two main categories of feature predicates: topological feature predicates, and non-terminal symbol feature predicates. Topological feature predicates evaluate whether a node with the value of its first arguments exists at some location in the parse tree generated from the second argument (e.g., *(is-left-child-of -3 -3x)*). Non-terminal symbol feature predicates are defined based on the non-terminal symbols used in the grammar rules. For example, *-3* is associated with the non-terminal symbol *SignedNumber* based on the grammar shown in Table 2.1. We plan to evaluate the quality of the automatically generated feature predicates in three domains fraction addition, algebra, and stoichiometry.

Related Work: Research on ILP (e.g., [74, 75, 89]) is closely related to this proposed work, as SimStudent uses FOIL as its "when" learner. ILP systems acquire logic programs that separate positive examples from negative ones given an encoding of the known background knowledge. Our approach differs from these systems in that it automatically generates the encoding based on a learned grammar, and calls an existing ILP algorithm to acquire the when part of the production rule.

Although there has been considerable work on representation change (e.g., [25, 43, 59, 66, 95]) in machine learning, little has used hierarchical representations to automatically create feature predicates. Furthermore, little has integrated the predicate learner into an intelligent agent. An exception comes from Li, Stracuzzi and Langley [57], where the predicate learning mechanism acquires feature predicates for an intelligent agent, Icarus, from its problem solving experience. But none of the above work uses learned representation hierarchies to create feature predicates as the proposed approach.

3.2 Comparing Deep Feature Learning+FOIL with Deep Learning as When-Learning

In the previous section, we proposed to use the deep feature learner and FOIL to learn when to fire a production rule. As FOIL is based on first-order logic, a rule is either applicable or not. In this framework, we decoupled this learning module into two components, an unsupervised statistical module that learns the world representation and generates a set of feature predicates, and a supervised logic-based module that uses the generated feature predicates to acquire the when part of the production rule. In comparison with this decoupled learning strategy, we would like to explore whether a joint model would perform better or not.

In order to make a fair comparison, we formulate the ***when*** part learning as a classification problem. For each production rule, we could train a classifier for it, where the input are the strings of input gathered from the ***where*** part of the production rule, and the output is whether the production rule should be fired or not.

We choose deep belief networks [31] as the model we would like to compare with, since it also focuses modeling complicated non-linear functions through learned features. Deep belief networks are probabilistic generative models that are composed of multiple layers of stochastic, latent variables. The latent variables typically have binary values and are often called hidden units or feature detectors. The deep belief networks will carry out an unsupervised learning process on strings of observations gathered by SimStudent, and

construct features for these representations. Then, a supervised learning process is followed where each production rule corresponds to a classifier. We plan to evaluate the effectiveness of both the proposed method and deep belief networks, and see which produces a better result.

3.3 Efficient Cross-Domain Learning of Complex Skills

We have mainly evaluated our approach in algebra, but the proposed approach is not limited to this domain. To evaluate the generality of the proposed approach and the effect of integration on prior knowledge, we plan to evaluate the proposed approach in other domains. There are four categories of domains we are currently considering: math domains (e.g., multi-column addition, percents, proportional reasoning), chemistry domains (e.g., stoichiometry, balancing chemical equations), second language learning (e.g., article selection), and software domains (e.g., using Excel to do basic statistical tests, automatic transformation of programs into a "standard" form). In the following part of this subsection, we will describe some of the above tasks in detail. In this thesis, we will select some tasks from two or three of the above categories, and evaluate the learning effectiveness of SimStudent.

3.3.1 Article Selection

Linguistic theory has long been adapted to the simplified assumption that knowledge of language is characterized by a categorical system of grammar. Nevertheless, many previous studies have shown that language users reliably and systematically make probabilistic syntactic choices from multi-dimensional information. While results are starting to accumulate [10], we have little by way of precise understanding of how the probabilistic syntactic choices are made, and how the knowledge is acquired. A probabilistic computational model of language learning that fits student learning data would be a significant achievement in theoretical integration within the learning sciences, and reveal insights on improving current education technologies.

Bresnan and Hay [10] took a first step towards modeling the probabilistic aspect of human language skills, and proposed a multivariate multilevel logistic regression model that can accurately predict the choices on unseen data in a recent study of English speakers' syntactic choices with give-type verbs during spontaneous conversations. Despite the promising results, a couple of limitations remain in the proposed approach. First, the proposed model relies on a set of strong (high-level) features (e.g. syntactic complexity, animacy) to achieve high prediction accuracy. Models based on such strong features will not be appropriate in capturing beginners' knowledge base. How to produce such features? Can it be automated? Second, as is well known, the process of making syntactic choices often requires reasoning based on logical relations among language features. For example, when choosing articles in English, one should use "the" either when the noun has already been mentioned, or when the noun is used with the word "same". However, the proposed logistic regression model, which makes predictions purely based on linear combinations of the given features, is unable to represent such logical relations among language features. This largely limits the range of tasks that could be modeled by the proposed model.

We plan to address both issues by exploring a particular combination of statistical and

logic-based learning: in particular, we use statistical deep feature learning to model how humans learn these language features, and combine this with logic-based learning to achieve human-like, rapid learning of article selection. One challenge we may face is that to acquire these language features, we need to capture both the syntax features and the semantic features. The current pCFG learner mainly captures the syntax of the language, but fails to capture the semantics of the language (e.g., water is an uncountable noun.). Instead of building a semantic parser (e.g., [101, 103]), which often requires additional annotations for learning, we plan to extend the pCFG learner to incorporate world knowledge that can be gathered from other information resources such as the web. For example, we can get knowledge such as "Amazon either refers to a website or a river" from webpages. But we don't know when Amazon means a website, and when it means a river. If we can acquire pCFG based on a sentence repository, we may be able to capture knowledge such as if Amazon is followed by a word "river", it is referring to a river 98% of time. By acquiring these knowledge, we would be able to build a rapid learning system of article selection.

3.3.2 Software Domains

One example domain is learning to perform basic statistical tests in Excel. Previous work [60, 80] has built intelligent tutors that teach students to perform tasks in Excel, but none has modeled how to learn to perform these tasks. We would like to demonstrate how users carry out basic statistical tests (e.g., t-test) in Excel to SimStudent, and let SimStudent learn it. To learn to perform this task, being able to model the two-dimensional displays correctly is an important component, which ties back to one of our previous proposed ideas. One key difference between Excel domains and previous tutoring domains is that previous tutoring domains often have fixed interface across problems in the same domain (e.g., equation solving), whereas the data in an Excel spreadsheet can be shown in any format (e.g., in rows or columns). The 2-D layout parser should be able to generate different parses for different datasets. Given different versions of such layout, instead of learning one production rule for each dataset layout, SimStudent should be able to capture a relatively small set of general rules that are able to carry out the desired statistical tests across multiple layouts.

Another example domain is automatic transformation of programs into a "standard form". When school teachers try to grade students' code for programs, it is often a challenging task as even if the students are all programming for the same algorithm, there are various ways to code it. Hence, teachers need to read through each different code and grade them separately. If there is a way to automatically transform different coding styles into a standard form (e.g., always using the while loop instead of the for loop), we could largely reduce the grading burden from the teacher. However, even manually coding such transformation can be challenging. In this task, we would like to demonstrate such transformation to SimStudent, and let it automatically learn such transformation rules. This domain is challenging since the format of a program can vary largely from case to case. Transforming different programs into a standard form is hard even for humans. The key to the success of such transformation is to get the correct parse of the program, i.e., the correct representation of the program. We plan to use the pCFG learner to acquire such representation, and then integrate it into SimStudent.

3.4 Better Understanding of Human Student Learning

Now that we have proposed to make various extensions to make SimStudent a more effective and general learner, the last direction we would like to explore is to use SimStudent to identify key features that differentiate fast learners from slow learners, and that produce more effective learning.

3.4.1 Automatic Student Model Discovery across Domains

We have shown that we can use SimStudent to discover human student models in algebra. We would like to test this in other domains (e.g., fraction addition, stoichiometry), and see whether we can use SimStudent to find better student models as well. Furthermore, we would like to train different versions of SimStudent with problems for human students, and match SimStudent's learning curve with human student. More specifically, for each human student, we plan to train SimStudent with the same problem sequence, and see whether SimStudent learns as fast as the human student.

3.4.2 Problem Order Implications for Learning Transfer

The third study we would like to carry out concerns with one of the most important variables that affects learning effectiveness, *the order of problems presented to students*. While most existing textbooks organize problems in a blocked order, in which all problems of one type (e.g. learning to solve equations of the form $S_1/V=S_2$) are completed before the student is switched to the next problem type, it is surprising that problems in an interleaved order often yields more effective learning. Numerous studies have experimentally demonstrated this effect (e.g., [13,22,26,33,51,84,85,102]). However, the cause of the effect is still unclear. A computational model that demonstrates such behavior would be a great help in better understanding this widely-observed phenomena, and might reveal insights that can improve current education technologies. We plan to conduct a controlled-simulation study using SimStudent. SimStudent will be trained on real-student problems that were of blocked orders or interleaved orders. We then plan to test whether the advantages of interleaved problem orders over blocked problem orders are exhibited across domains. After that, we plan to check the causes of such effect by inspecting SimStudent's learning processes and learning outcomes, which are not easily obtainable from human subjects.

3.4.3 Prior Knowledge vs. Error Types

Furthermore, we would also like to see what causes human students in making certain types of errors. Previous studies [61] have shown that SimStudent is able to produce more human-like errors given fewer domain-specific operator functions. To further understand how other types of prior knowledge affect learning behavior, we will first classify common types of errors made by human students. Then, we will manipulate SimStudent's prior knowledge including the acquired deep features based on different training curricula, and feature predicates, and test to see lacking what kind of prior knowledge would cause certain types of errors.

4 Schedule

Year	Month	Research Activity
2012	Apr-May	Comparing deep feature learning+FOIL with deep learning as when-learning
	Jun	Learning to perceive two-dimensional displays using probabilistic grammars
	Jul	Creating features from a learned grammar in SimStudent
	Aug	Automatic student model discovery in multiple domains
	Sep	Experiment with prior knowledge vs. error types
	Oct	Problem order implications for learning transfer
	Nov-Dec	Learning by tutoring in a software domain
2013	Jan-Feb	Learning by tutoring in a second language learning domain
	Mar-Apr	Write thesis
	May	Defend thesis

Table 4.1: Schedule towards the completion of the thesis. Our plan is to defend in May 2013.

References

[1] John R. Anderson. *Rules of the Mind*. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1993.

[2] John R. Anderson and Ross Thompson. Similarity and analogical reasoning. chapter Use of analogy in a production system architecture, pages 267–297. Cambridge University Press, New York, NY, USA, 1989.

[3] Yuichiro Anzai and Herbert A Simon. The theory of learning by doing. *Psychological Review*, 86(2):124–140, 1979.

[4] Arvind Arasu and Hector Garcia-Molina. Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 337–348, New York, NY, USA, 2003. ACM.

[5] Paul Baffes and Raymond Mooney. Refinement-based student modeling and automated bug library construction. *Journal of Artificial Intelligence in Education*, 7(1):75–116, 1996.

[6] Tiffany Barnes. The Q-matrix method: Mining student response data for knowledge. In *Proceedings AAAI Workshop Educational Data Mining*, pages 1–8, Pittsburgh, PA, 2005.

[7] Yoshua Bengio. Learning deep architectures for ai. *Foundations Trends in Machine Learning*, 2:1–127, January 2009.

[8] Yoshua Bengio, Olivier Delalleau, and Clarence Simard. Decision trees do not generalize to new variations. *Computational Intelligence*, 26(4):449–467, November 2010.

[9] David Blei and Jon McAuliffe. Supervised topic models. In *Proceedings of the Twenty-Fifth Annual Conference on Neural Information Processing Systems*, pages 121–128, Cambridge, MA, 2007. MIT Press.

[10] Joan Bresnan and Jennifer Hay. Gradient grammar: An effect of animacy on the syntax of give in New Zealand and American English. *Lingua*, 118(2):245–259, 2008.

[11] Richard R. Burton. Diagnosing bugs in a simple procedural skill. In *Intelligent Tutoring Systems*, pages 157–184. Academic Press, 1982.

[12] Michael J. Cafarella, Alon Y. Halevy, Daisy Z. Wang, Eugene W. 0002, and Yang Zhang. Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment*, 1(1):538–549, 2008.

[13] H Carnahan, D L Van Eerd, and F Allard. A note on the relationship between task requirements and the contextual interference effect. *Journal of Motor Behavior*, 22(1):159–169, 1990.

[14] Hao Cen, Kenneth Koedinger, and Brian Junker. Learning factors analysis - a general method for cognitive model evaluation and improvement. In *Proceedings of the 8th International Conference on Intelligent Tutoring Systems*, pages 164–175, 2006.

[15] Tak-Wai Chan and Chih-Yueh Chou. Exploring the design of computer supports for reciprocal tutoring. *International Journal of Artificial Intelligence in Education*, 8:1–29, 1997.

[16] William G. Chase and Herbert A. Simon. Perception in chess. *Cognitive Psychology*, 4(1):55–81, January 1973.

[17] Michelene T. H. Chi, Paul J. Feltovich, and Robert Glaser. Categorization and representation of physics problems by experts and novices. *Cognitive Science*, 5(2):121–152, June 1981.

[18] P. A. Chou. Recognition of Equations Using a Two-Dimensional Stochastic Context-Free Grammar. In *Proceedings of Visual Communications and Image Processing*, volume 1199, pages 852–863, November 1989.

[19] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 109–118, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[20] Allen Cypher, Daniel C. Halbert, David Kurlander, Henry Lieberman, David Maulsby, Brad A. Myers, and Alan Turransky, editors. *Watch what I do: programming by demonstration.* MIT Press, Cambridge, MA, 1993.

[21] Peter Dayan, Geoffrey E. Hinton, Radford M. Neal, and Richard S. Zemel. The Helmholtz Machine. *Neural Computation*, 7(5):889–904, December 1995.

[22] P Del Rey. Effects of contextual interference on the memory of older females differing in levels of physical activity. *Perceptual and motor skills*, 55(1):171–180, 1982.

[23] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[24] Hazem Elmeleegy, Jayant Madhavan, and Alon Halevy. Harvesting relational tables from lists on the web. *The VLDB Journal*, 20:209–226, 2011.

[25] Tom Fawcett. Knowledge-based feature discovery for evaluation functions. *Computational Intelligence*, 12(1), 1996.

[26] T E Gabriele, C R Hall, and E E Buckolz. Practice schedule effects on the acquisition and retention of a motor skill. *Human Movement Science*, 6:1–16, 1987.

[27] Fernand Gobet. Chunking models of expertise: implications for education. *Applied Cognitive Psychology,*, 19(3):183–204, January 2005.

[28] Fernand Gobet and Herbert A. Simon. Five seconds or sixty? presentation time in expert memory. *Cognitive Science*, 24(4):651–682, 2000.

[29] G. E. Hinton. To recognize shapes, first learn to generate images. *Progress in brain research*, 165:535–547, 2007.

[30] G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal. The "wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161, May 1995.

[31] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[32] Rebecca Hwa. Supervised grammar induction using training data with limited constituent information. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 73–79, Stroudsburg, PA, USA, 1999. Association for Computational Linguistics.

[33] Otto Jelsma and Jules M Pieters. Practice schedule and cognitive style interaction in learning a maze task. *Applied Cognitive Psychology*, 3(1):73–83, 1989.

[34] Charles Kemp and Joshua B B. Tenenbaum. The discovery of structural form. *Proceedings of the National Academy of Sciences of the United States of America*, July 2008.

[35] Charles Kemp and Fei Xu. An ideal observer model of infant object perception. In Daphne Koller, Dale Schuurmans, Yoshua Bengio, and Léon Bottou, editors, *NIPS*, pages 825–832. MIT Press, 2008.

[36] Kenneth R. Koedinger and John R. Anderson. Abstract Planning and Perceptual Chunks: Elements of Expertise in Geometry. *Cognitive Science*, 14:511–550, 1990.

[37] Kenneth R. Koedinger, Ryan S.J.d. Baker, Kyle Cunningham, Alida Skogsholm, Brett Leber, and John Stamper. A data repository for the EDM community: The PSLC DataShop, 2010.

[38] Kenneth R. Koedinger and Albert Corbett. Cognitive Tutors: Technology Bringing Learning Sciences to the Classroom. pages 60–77. Cambridge University Press, Cambridge, 2006.

[39] Kenneth R. Koedinger and Elizabeth A. McLaughlin. Seeing language learning inside the math: Cognitive analysis yields transfer. In *Proceedings of the 32nd Annual Conference of the Cognitive Science Society*, pages 471–476, Austin, TX, 2010.

[40] Kenneth R. Koedinger and Mitchell J. Nathan. The real story behind story problems: Effects of representations on quantitative reasoning. *The Journal of Learning Sciences*, 13(2):129–164, 2004.

[41] John E. Laird, Allen Newell, and Paul S. Rosenbloom. Soar: an architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.

[42] John E. Laird, Paul S. Rosenbloom, and Allen Newell. Chunking in soar: The anatomy of a general learning mechanism. *Machine Learning*, 1:11–46, 1986.

[43] Pat Langley, Gary L. Bradshaw, and Herbert A. Simon. Rediscovering chemistry with the BACON system. volume 2, pages 307–329. Morgan Kaufmann, San Mateo, California, 1986.

[44] Pat Langley and Dongkyu Choi. A unified cognitive architecture for physical agents. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, Boston, 2006.

[45] Pat Langley and Stellan Ohlsson. Automated cognitive modeling. In *Proceedings of the Fourth National Conference on Artificial Intelligence*, pages 193–197, Austin, TX, 1984. Morgan Kaufmann.

[46] Pat Langley and Sean Stromsten. Learning context-free grammars with a simplicity bias. In *Proceedings of the 11th European Conference on Machine Learning*, pages 220–228, London, UK, 2000. Springer-Verlag.

[47] K. Lari and S. J. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56, 1990.

[48] Tessa Lau and Daniel S. Weld. Programming by demonstration: An inductive learning formulation. In *Proceedings of the 1999 international conference on intelligence user interfaces*, pages 145–152, 1998.

[49] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1:541–551, December 1989.

[50] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[51] Timothy D Lee and Richard A Magill. The locus of contextual interference in motor-skill acquisition. *Journal Of Experimental Psychology. Learning Memory And Cognition*, 9(4):730–746, 1983.

[52] Nan Li, William W. Cohen, and Kenneth R. Koedinger. A computational model of accelerated future learning through feature recognition. In *ITS'10: Proceedings of 10th International Conference on Intelligent Tutoring Systems*, pages 368–370, 2010.

[53] Nan Li, William W. Cohen, and Kenneth R. Koedinger. Efficient cross-domain learning of complex skills. In *Proceedings of the 11th International Conference on Intelligent Tutoring Systems*, 2012.

[54] Nan Li, William W. Cohen, and Kenneth R. Koedinger. Integrating representation learning and skill learning in a human-like intelligent agent. Technical Report CMU-MLD-12-1001, Carnegie Mellon University, January 2012.

[55] Nan Li, William W. Cohen, Noboru Matsuda, and Kenneth R. Koedinger. A machine learning approach for automatic student model discovery. In *Proceedings of the 4th International Conference on Educational Data Mining*, pages 31–40, 2011.

[56] Nan Li, Subbarao Kambhampati, and Sungwook Yoon. Learning probabilistic hierarchical task networks to capture user preferences. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, Pasadena, CA, 2009.

[57] Nan Li, David J. Stracuzzi, and Pat Langley. Learning conceptual predicates for teleoreactive logic programs. In *Proceedings of the Eighteenth International Conference on Inductive Logic Programming: Late-Breaking Papers*, Prague, Czech Republic, 2008.

[58] Jayant Madhavan, David Ko, Lucja Kot, Vignesh Ganapathy, Alex Rasmussen, and Alon Halevy. Google's deep web crawl. *Proceedings VLDB Endowment*, 1(2):1241–1252, August 2008.

[59] Mario Martín and Hector Geffner. Learning generalized policies from planning examples using concept languages. *Applied Intelligence*, 20:9–19, January 2004.

[60] Santosh Mathan. *Recasting the feedback debate: benefits of tutoring error detection and correction skills*. PhD thesis, Pittsburgh, PA, USA, 2003.

[61] Noboru Matsuda, Andrew Lee, William W. Cohen, and Kenneth R. Koedinger. A computational model of how learner errors arise from weak prior knowledge. In *Proceedings of Conference of the Cognitive Science Society*, 2009.

[62] Matthew Michelson and Craig A. Knoblock. Unsupervised information extraction from unstructured, ungrammatical data sources on the world wide web. *International Journal on Document Analysis and Recognition*, 10(3):211–226, December 2007.

[63] Tom Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1982.

[64] Tom M. Mitchell, Sridhar Mahadevan, and Louis I. Steinberg. Leap: a learning apprentice for vlsi design. In *Proceedings of the 9th international joint conference on Artificial intelligence*, pages 573–580, San Francisco, CA, 1985.

[65] Raymond J. Mooney. *A General Explanation-Based Learning Mechanism and its Application to Narrative Understanding*. Morgan Kaufmann, San Mateo, CA, 1990.

[66] S. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 339–352. Morgan Kaufmann, 1988.

[67] Stephen Muggleton and Luc de Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19:629–679, 1994.

[68] David M. Neves. Learning procedures from examples and by doing. In *Proceedings of the 9th international joint conference on Artificial intelligence - Volume 1*, pages 624–630, San Francisco, CA, USA, 1985. Morgan Kaufmann Publishers Inc.

[69] Allen Newell. The knowledge level. *Artificial Intelligence*, 18(1):87–127, 1982.

[70] Alexandru Niculescu-Mizil and Rich Caruana. Inductive transfer for bayesian network structure learning. In *Proceedings of the 11th International Conference on AI and Statistics*, 2007.

[71] Stellan Ohlsson. *Computational Models of Skill Acquisition*, chapter 13, pages 359–395. Cambridge University Press, 2008.

[72] Philip I. Pavlik, Hao Cen, and Kenneth R. Koedinger. Learning Factors Transfer Analysis: Using Learning Curve Analysis to Automatically Generate Domain Models. In *Proceedings of 2nd International Conference on Educational Data Mining*, pages 121–130, 2009.

[73] Timo Niemirepo Pentti Hietala. The competence of learning companion agents. *International Journal of Artificial Intelligence in Education*, 9:178–âĂŞ192, 1998.

[74] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990.

[75] Luc De Raedt and Luc Dehaspe. Clausal discovery. *Machine Learning*, 26(2):99–146, 1997.

[76] Rajat Raina, Andrew Y. Ng, and Daphne Koller. Constructing informative priors using transfer learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 713–720, New York, NY, 2006.

[77] Marc'Aurelio Ranzato, Fu J. Huang, Y. Lan Boureau, and Yann LeCun. Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 0:1–8, 2007.

[78] Matthew Richardson and Pedro Domingos. Markov logic networks. *Mach. Learn.*, 62(1-2):107–136, 2006.

[79] Howard B. Richman, James J. Staszewski, and Herbert A. Simon. Simulation of expert memory using EPAM IV. *Psychological Review*, pages 305–330, 1995.

[80] Steven Ritter and Kenneth R. Koedinger. An architecture for plug-in tutor agents. *Journal of Artificial Intelligence in Education*, 7(3-4):315–347, January 1996.

[81] Brian Roark and Michiel Bacchiani. Supervised and unsupervised pcfg adaptation to novel domains. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 126–133, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[82] Lawrence K. Saul, Tommi Jaakkola, and Michael I. Jordan. Mean field theory for sigmoid belief networks. *Journal of Artificial Intelligence Research*, 4:61–76, 1996.

[83] Alberto Segre. A learning apprentice system for mechanical assembly. In *Proceedings of the Third IEEE Conference on AI for Applications*, pages 112–117, 1987.

[84] H Sekiya, R A Magill, and D I Anderson. The contextual interference effect in parameter modifications of the same generalized motor program. *Research quarterly for exercise and sport*, 67(1):59–68, 1996.

[85] John B Shea and Robyn L Morgan. Contextual interference effects on the acquisition, retention, and transfer of a motor skill. *Journal of Experimental Psychology Human Learning Memory*, 5(2):179–187, 1979.

[86] Patrice Y. Simard, Dave Steinkraus, and John C. Platt. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. In *ICDAR '03: Proceedings of the Seventh International Conference on Document Analysis and Recognition*, Washington, DC, USA, 2003. IEEE Computer Society.

[87] J. M. Siskind, Jr J. Sherman, I. Pollak, M. P. Harper, and C. A. Bouman. Spatial random tree grammars for modeling hierarchal structure in images with regions of arbitrary shape. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29:1504–1519, 2007.

[88] D. H. Sleeman and M. J Smith. Modeling students' problem solving. *Artificial Intelligence*, 16:171–187, 1981.

[89] A. Srinivasan. *The Aleph Manual*, 2004.

[90] Andreas Stolcke. *Bayesian learning of probabilistic language models*. PhD thesis, Berkeley, CA, USA, 1994.

[91] Niels A. Taatgen and Frank J. Lee. Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, 45(1):61–75, 2003.

[92] Kikumi K. Tatsuoka. Rule space: An approach for dealing with misconceptions based on item response theory. *Journal of Educational Measurement*, pages 345–354, 1983.

[93] Ivan Titov and James Henderson. Constituent Parsing with Incremental Sigmoid Belief Networks. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 632–639, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[94] L. Torrey, J. Shavlik, T. Walker, and R. Maclin. Relational macros for transfer in reinforcement learning. In *Proceedings of the 17th Conference on Inductive Logic Programming*, Corvallis, Oregon, 2007.

[95] Paul E. Utgoff. *Shift of Bias for Inductive Concept Learning.* PhD thesis, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1984.

[96] Kurt VanLehn. *Mind Bugs: The Origins of Procedural Misconceptions.* MIT Press, Cambridge, MA, USA, 1990.

[97] Kurt Vanlehn and William Ball. A version space approach to learning context-free grammars. *Machine Learning*, 2(1):39–74, March 1987.

[98] Kurt Vanlehn, Stellan Ohlsson, and Rod Nason. Applications of simulated students: an exploration. *Journal of Artificial Intelligence in Education*, 5:135–175, February 1994.

[99] Michael Villano. Probabilistic student models: Bayesian belief networks and knowledge space theory. In *Proceedings of the 2nd International Conference on Intelligent Tutoring Systems*, pages 491–498, Heidelberg, 1992.

[100] J. G. Wolff. Language acquisition, data compression and generalization. *Language and Communication*, 2:57–89, 1982.

[101] Yuk Wah Wong and Raymond J. Mooney. Learning for semantic parsing with statistical machine translation. In *Proceedings of Human Language Technology Conference / North American Chapter of the Association for Computational Linguistics Annual Meeting (HLT-NAACL-06)*, pages 439–446, New York City, NY, 2006.

[102] D E Young, M J Cohen, and W S Husak. Contextual interference and motor skill acquisition: On the processes that influence retention. *Human Movement Science*, 12(5):577–600, 1993.

[103] Luke S. Zettlemoyer and Michael Collins. Learning context-dependent mappings from sentences to logical form. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - Volume 2*, ACL '09, pages 976–984, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[104] Yanhong Zhai and Bing Liu. Extracting web data using instance-based learning. *World Wide Web*, 10(2):113–132, June 2007.